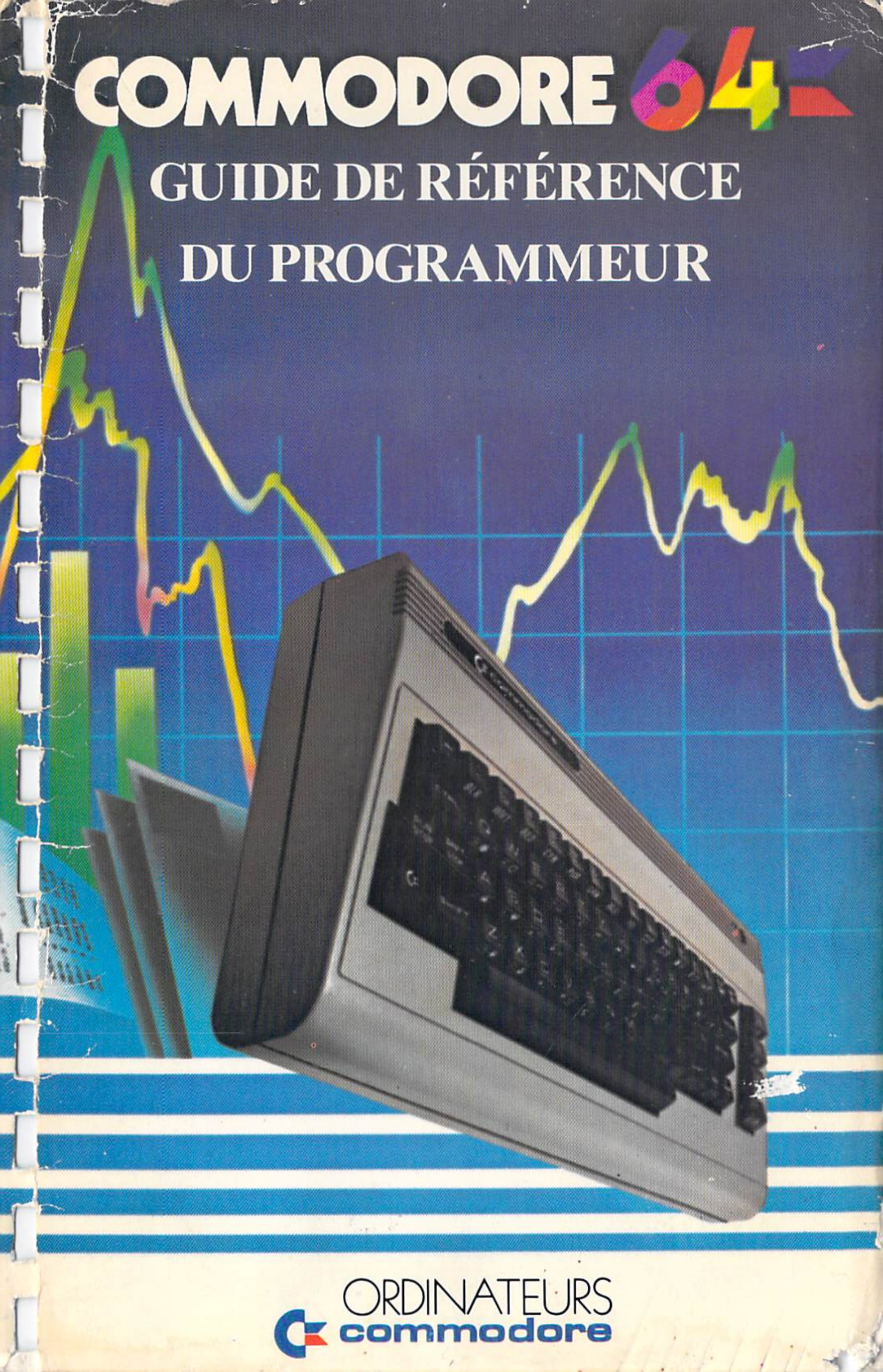

COMMODORE 64
GUIDE DE RÉFÉRENCE
DU PROGRAMMEUR
ORDINATEURS
commodore

GUIDE DE RÉFÉRENCE DU PROGRAMMEUR DE COMMODORE 64

Publié par
Commodore Business Machines, Inc.
et par
Howard W. Sams & Co., Inc.

PREMIÈRE ÉDITION
QUATRIÈME IMPRESSION—1983

TABLE DES MATIÈRES

INTRODUCTION

Le **GUIDE DE RÉFÉRENCE DU PROGRAMMEUR DE COMMODORE 64** doit servir d'outil de travail et de source de référence en vue de tirer le maximum des possibilités inhérentes du **COMMODORE 64.** Ce manuel contient les informations nécessaires aux programmes, de l'exemple le plus simple au plus complexe. Le **GUIDE DE RÉFÉRENCE DU PROGRAMMEUR** s'adresse au programmeur débutant en BASIC ou au professionnel expérimenté en langage machine 6502; tous peuvent en tirer des informations pour mettre au point des programmes originaux. Par la même occasion, ce manuel montre l'étendue des possibilités du **COMMODORE 64.**

Le **GUIDE DE RÉFÉRENCE** ne vise pas à apprendre le langage de programmation BASIC ou le langage machine 6502. Il présente cependant un lexique complet des termes et une méthode "semi-pédagogique" pour plusieurs sections du manuel. Si l'on ne possède pas encore une connaissance pratique du BASIC et de son utilisation en programmation, nous recommandons d'étudier le **GUIDE DE L'UTILISATEUR DU COMMODORE 64** fourni avec l'ordinateur. Le **GUIDE DE L'UTILISATEUR** présente une introduction simple au langage de programmation BASIC. Si l'on ne comprend pas encore bien le BASIC, passer alors à la fin de ce manuel (ou à l'annexe N du **GUIDE DE L'UTILISATEUR**) et consulter la bibliographie.

Le **GUIDE DE RÉFÉRENCE DU PROGRAMMEUR DE COMMODORE 64** n'est rien d'autre qu'un ouvrage de référence. Comme la plupart des livres de ce genre, les possibilités d'applications originales des informations qu'il contient dépendent des connaissances préalables que l'on a du sujet. Si l'on n'est qu'un programmeur débutant, on ne sera pas en mesure d'appliquer les indications et les données de ce guide tant que l'on ne possédera pas une connaissance plus étendue de la programmation.

Ce guide met à la disposition de l'utilisateur une mine de précieux renseignements de programmation rédigés en langage courant, tout en expliquant le jargon de la programmation. Par ailleurs, le professionnel y trouvera tous les renseignements nécessaires pour mettre à contribution les possibilités du **COMMODORE 64.**

CONTENU DU GUIDE

- Notre répertoire BASIC complet comprend les commandes, instructions et fonctions du langage BASIC de Commodore, indiquées par ordre alphabétique. Nous avons mis sur pied une liste abrégée de tous les mots et de leurs abréviations. Elle est suivie d'une section donnant une définition plus précise de chaque mot avec des exemples de programme BASIC qui illustrent leur emploi.
- Notre aperçu est destiné à mettre le programmeur sur la voie de l'utilisation du langage machine avec les programmes en BASIC.
- Le KERNAL est une puissante caractéristique de tous les ordinateurs Commodore. Grâce à ce dispositif, les programmes rédigés aujourd'hui peuvent encore servir avec un futur modèle d'ordinateur Commodore.
- La section de programmation d'entrée/sortie donne la possibilité de pousser l'ordinateur jusqu'à ses limites. Elle explique les raccordements et l'utilisation des stylos lumineux et manettes de commande avec les unités de disques, imprimantes et dispositifs de télécommunications ou *modems*.
- On peut aborder l'univers des caractères programmables et des graphiques à haute définition qui permettent d'arriver à des images animées parmi les plus évoluées et les plus détaillées de la micro-informatique.
- On peut aussi aborder la synthèse de la musique et créer des airs et des effets sonores grâce au meilleur synthétiseur intégré à ce jour à un ordinateur individuel.
- Pour le spécialiste de la programmation, la section de langage programmable renseigne sur les possibilités d'exécution du **COMMODORE 64** avec le CP/M* et les langages évolués qui s'ajoutent au BASIC.

Le **GUIDE DE RÉFÉRENCE DU PROGRAMMEUR DE COMMODORE 64** constitue un outil pratique qui doit procurer de nombreuses heures agréables de programmation.

---

*CP/M est une marque déposée de Digital Research, Inc.

UTILISATION DU GUIDE DE RÉFÉRENCE

Dans ce manuel, nous utilisons certaines notations conventionnelles pour représenter la syntaxe (structure des phrases de programmation) des commandes ou instructions en BASIC et pour indiquer les parties requises et facultatives de chaque mot clé en BASIC. Les règles à utiliser dans l'interprétation de la syntaxe des instructions sont les suivantes:

1. Les mots clés en BASIC sont indiqués en majuscules. Ils doivent apparaître aux points spécifiés dans l'instruction, être introduits et orthographiés exactement comme il est indiqué.
2. Les éléments mis entre guillemets (" ") indiquent les données variables que l'on doit introduire. Les guillemets et les données entre ces guillemets doivent apparaître aux points indiqués dans chaque instruction.
3. Les éléments entre crochets ([ ]) indiquent un paramètre facultatif d'instruction. Un paramètre sert de limite ou de qualification supplémentaire aux instructions. Si l'on utilise un paramètre facultatif, on doit indiquer les données qui lui correspondent. En outre, les points de suspension (. . .) indiquent que l'on peut répéter un élément facultatif autant de fois que le permet une ligne de programmation.
4. Si un élément entre crochets ([ ]) est SOULIGNÉ, il faut alors utiliser les caractères précisés dans les paramètres facultatifs; il faut aussi les orthographier comme il est indiqué.
5. Les éléments entre les signes inférieur à et supérieur à (< >) indiquent des données variables que l'on doit préciser. L'oblique (/) indique que l'on doit choisir entre deux options qui s'excluent mutuellement.

EXEMPLE DE FORMAT DE SYNTAXE:

OPEN <numéro de fichier>, <dispositif> [, <adresse>], ["<unité>:<nom de fichier>] [, <mode>]"

EXEMPLES D'INSTRUCTIONS RÉELLES:

10 OPEN 2,8,6,"0:FEUILLE DE STOCK,S,W"
20 OPEN 1,1,2,"CHEQUIER"
30 OPEN 3,4

Quand on applique réellement les conventions de syntaxe dans une situation pratique, la séquence des paramètres des instructions peut ne pas correspondre exactement à celle indiquée dans les exemples de syntaxe. Les exemples ne visent pas à montrer chaque séquence possible; ils ne cherchent qu'à présenter tous les paramètres requis et facultatifs.

Les exemples de programmation de cet ouvrage sont indiqués avec des espaces vierges séparant les mots et les opérateurs afin d'être plus lisibles. En BASIC, il n'y a normalement pas besoin d'espace entre les mots mais leur omission peut conduire à une syntaxe ambiguë ou incorrecte.

Nous donnons ci-dessous des exemples et descriptions des symboles utilisés pour différents paramètres d'instruction dans les chapitres qui suivent. La liste n'indique pas toutes les possibilités, mais elle donne une meilleure idée de la présentation des exemples de syntaxe.

SYMBOLE	EXEMPLE	DESCRIPTION
<numéro de fichier>	**50**	Numéro de fichier logique
<dispositif>	**4**	Numéro de dispositif de matériel
<adresse>	**15**	Numéro d'adresse de dispositif secondaire de bus série
<unité>	**0**	Numéro d'unité de disques physique
<nom de fichier>	**"DONNEES D'ESSAI"**	Nom d'un fichier de données ou de programme
<constante>	**"ABCDEFG"**	Données littérales fournies par le programmeur
<variable>	**X145**	Tout nom ou constante de variable de données BASIC
<chaîne>	**AB$**	Utilisation requise d'une variable en chaîne
<numéro>	**12345**	Utilisation requise d'une variable numérique
<numéro de ligne>	**1000**	Numéro réel de ligne de programme
<numérique>	**1.5E4**	Variable entière ou à point décimal flottant

GUIDE DES APPLICATIONS DU COMMODORE 64

Au moment d'acheter un ordinateur, on s'est probablement interrogé sur l'usage que l'on pourrait en faire et sur ses applications.

Le **COMMODORE 64** présente l'énorme avantage de faire simplement ce qu'on veut lui faire faire! Il peut calculer le budget d'un foyer ou d'une entreprise et suivre les dépenses. Il peut s'utiliser pour le traitement de texte ou avec les jeux électroniques. Il peut même chanter. On peut aussi lui faire créer des dessins animés . . . la liste est pratiquement infinie. S'il servait simplement à remplir l'une des fonctions que nous venons de citer, le prix du **COMMODORE 64** serait amplement justifié. Le **COMMODORE 64** est cependant un outil d'ordinateur complet qui peut s'acquitter de TOUTES ces tâches et de quelques autres par-dessus le marché!

En dehors de toutes les fonctions que nous venons d'énumérer, on peut trouver nombre d'autres idées nouvelles et pratiques en s'inscrivant à un club local d'utilisateur **Commodore,** en s'abonnant aux magazines **COMMODORE** et **POWER/PLAY** et en se raccordant au réseau d'informations **COMMODORE** sur le CompuServe™.

APPLICATION	REMARQUES/CONDITIONS
JEUX ÉLECTRONIQUES	On dispose des jeux Bally Midway comme la Course Omega, Gorf et le Sorcier de Wor, ainsi que des jeux éducatifs comme le Prof de maths I, le Gardien d'enfants et l'Artiste Commodore.
PUBLICITÉ ET COMMERCIALISATION	Raccorder le **COMMODORE 64** à un téléviseur, installez-le dans une vitrine avec un message clignotant, animé et musical et vous disposez d'une excellente présentation de lieu de vente.
ANIMATION	Les caractères graphiques programmables du Commodore permettent de créer de véritables dessins animés avec 8 niveaux différents de façon que les formes peuvent se déplacer devant ou derrière les unes les autres.
GARDIEN D'ENFANTS	La cartouche de GARDIEN D'ENFANTS du **COMMODORE 64** peut occuper les tout-petits pendant des heures tout en leur apprenant l'alphabet et en les familiarisant avec le clavier. Il leur apprend aussi des relations et des concepts spéciaux.
PROGRAMMATION DE BASE	Le **GUIDE DE L'UTILISATEUR DU COMMODORE 64** et la série d'ouvrages et de cassettes d'enseignement progressif de la programmation constituent un excellent point de départ.
FEUILLE DE TRAVAIL DE GESTION	Le **COMMODORE 64** peut s'utiliser avec la série "Easy" d'auxiliaires de gestion qui comprennent les plus puissants programmes de traitement de texte et la plus grande feuille de travail offerts avec un ordinateur individuel.
COMMUNICATION	Pour aborder le monde captivant de la gestion des réseaux d'ordinateurs. En raccordant un VICMODEM au **COMMODORE 64**, on peut communiquer avec d'autres possesseurs d'ordinateur, dans le monde entier.

Par ailleurs, si l'on se joint au **RÉSEAU D'INFORMATIONS COMMODORE** sur le CompuServe™, on peut bénéficier des toutes dernières nouvelles et mises à jour sur tous les produits Commodore, les nouvelles financières, les services d'achat à domicile. On peut aussi participer à des jeux avec les amis que l'on rencontre dans les systèmes d'informations auxquels on participe.

COMPOSITION MUSICALE

Le **COMMODORE 64** est équipé d'un synthétiseur musical intégré particulièrement perfectionné. Il possède trois voix totalement programmables, neuf octaves musicales intégrales et quatre formes d'onde contrôlables. Les cartouches et les ouvrages de musique Commodore facilitent la reproduction de tous les types de musique et d'effets sonores ainsi que leur création.

CP/M*

Commodore offre un complément CP/M* et l'accès au logiciel par une cartouche facile à charger.

AMÉLIORATION DE LA DEXTÉRITÉ

Plusieurs jeux Commodore, comme l'Atterrissage sur Jupiter et la simulation de conduite de nuit développent la dextérité manuelle et la coordination des yeux et des mains.

ENSEIGNEMENT

L'utilisation d'un ordinateur est elle-même une activité pédagogique, mais le manuel de ressources d'enseignement COMMODORE présente des renseignements généraux sur les utilisations éducatives des ordinateurs. Nous offrons aussi un choix de cartouches d'enseignement permettant de se familiariser avec des domaines comme la musique et les maths en passant par les arts et l'astronomie.

LANGUES ÉTRANGÈRES

Le jeu de caractères programmables du **COMMODORE 64** permet de remplacer le jeu de caractères normaux par les signes et caractères d'une langue étrangère, définis par l'utilisateur.

---

*CP/M est une marque déposée de Digital Research, Inc.

GRAPHIQUES ET ART

En plus des caractères graphiques programmables cités ci-dessus, le **COMMODORE 64** permet le tracé de graphiques multicolores à haute définition, les caractères programmables et les combinaisons de tous les modes d'affichage différents de graphiques et de caractères.

COMMANDE D'INSTRUMENT

Le **COMMODORE 64** est doté d'un accès série, d'un accès RS-232 et d'un accès d'utilisateur destiné à différentes applications industrielles spéciales. En supplément facultatif, on peut aussi disposer d'une cartouche IEEE/488.

REVUES ET RÉDACTIONS CRÉATRICES

Le **COMMODORE 64** permettra bientôt l'utilisation d'un système exceptionnel de traitement de texte avec des possibilités comparables ou supérieures à celles des appareils de traitement de texte parmi les plus coûteux sur le marché. On peut évidemment conserver les informations sur une unité de disques 1541 ou un magnétocassette Datassette™ et les imprimer à l'aide d'un TRACEUR ou d'une IMPRIMANTE VIC.

CRAYON LUMINEUX

On peut exécuter les applications nécessitant ce genre d'accessoire à l'aide d'un crayon lumineux branché dans le raccord d'accès de jeu du **COMMODORE 64.**

PROGRAMMATION EN CODE MACHINE

Le **GUIDE DE RÉFÉRENCE DU PROGRAMMEUR DE COMMODORE 64** comprend une section en langage machine ainsi qu'une section d'interface BASIC/code machine. La bibliographie permet en outre une étude plus poussée.

IMPRESSION DE LA FEUILLE DE PAIE ET DES FORMULES

On peut programmer le **COMMODORE 64** pour qu'il s'accommode de différentes applications de gestion avec introduction des informations. Les majuscules et minuscules combinées aux graphiques de gestion **C64** facilitent la préparation de formules que l'on peut ensuite sortir sur imprimante.

IMPRESSION

Le **COMMODORE 64** permet l'interface avec des traceurs et des imprimantes à matrice de points de type machine à écrire.

RECETTES DE CUISINE — Le **COMMODORE 64** permet de mémoriser des recettes de cuisine sur disque ou sur cassette; on élimine ainsi les cartes qui deviennent introuvables quand on en a besoin.

SIMULATIONS — Les simulations par ordinateur permettent de mener des expériences dangereuses ou coûteuses avec le minimum de risques et de frais.

DONNÉES SUR LES SPORTS — Les services Source™ et CompuServe™ offrent des informations sportives que l'on peut mettre à profit avec le **COMMODORE 64** et un VICMODEM.

COTES BOURSIÈRES — Avec un VICMODEM et un abonnement aux services des réseaux correspondants, le **COMMODORE 64** permet de disposer des dernières informations boursières.

Le **COMMODORE 64** permet d'accéder à de nombreuses autres applications. Pour le travail ou les loisirs, à la maison, à l'école ou au bureau, le **COMMODORE 64** apporte une solution pratique à la quasi-totalité des problèmes quotidiens.

Commodore désire que l'utilisateur sache que son soutien ne fait que **COMMENCER** à l'achat d'un de ses ordinateurs. Pour cette raison, nous avons mis sur pied deux publications avec des informations Commodore venant du monde entier et un réseau bidirectionnel de données informatiques particulièrement précieux pour les utilisateurs canadiens et américains.

En outre, nous encourageons et soutenons sans réserve l'expansion des clubs d'utilisateurs Commodore dans le monde entier. Ils constituent d'excellentes sources de renseignements pour les possesseurs d'ordinateur Commodore, du débutant au spécialiste confirmé. Les magazines et le réseau, que nous étudions plus en détail ci-dessous, fournissent des renseignements à jour sur le club d'utilisateurs d'une région donnée.

Enfin, le détaillant Commodore local constitue une source précieuse de soutien et de renseignements sur les ordinateurs Commodore.

POWER/PLAY
Le magazine de l'ordinateur individuel

Au niveau des loisirs, de l'enseignement et des applications pratiques à domicile, **POWER/PLAY** constitue LA source essentielle de renseignements pour les utilisateurs résidentiels d'un ordinateur Commodore. Localiser les clubs locaux et déterminer leurs activités pour se familiariser sur le logiciel, les jeux, les techniques de programmation, les télécommunications et les produits nouveaux. **POWER/PLAY** relie individuellement

les utilisateurs, les réalisateurs extérieurs de matériel et de logiciels et la compagnie Commodore proprement dite. La publication est trimestrielle. L'abonnement annuel à ce magazine passionnant se monte à $10.

COMMODORE
Le magazine de la micro-informatique

Consulté par les enseignants, les hommes d'affaires et les étudiants ainsi que par les informaticiens amateurs, **COMMODORE Magazine** est le lien essentiel pour le partage d'informations exclusives sur les applications plus techniques du système Commodore. Les rubriques courantes traitent des affaires, des sciences et de l'enseignement, des conseils de programmation, d'extraits d'un bloc-notes technique et de nombreux autres articles intéressant l'utilisateur ou l'acheteur éventuel d'un équipement Commodore pour des applications commerciales, scientifiques ou pédagogiques. **COMMODORE** complète parfaitement le magazine **POWER/PLAY.** L'abonnement annuel est de $15 pour six numéros.

POUR EN SAVOIR ENCORE PLUS . . .
. . . APPELER NOTRE MAGAZINE "SANS PAPIER" AU SERVICE DES UTILISATEURS

RÉSEAU D'INFORMATIONS COMMODORE

Le magazine de l'avenir est arrivé. Pour compléter les magazines **POWER/PLAY** et **COMMODORE,** le **RÉSEAU D'INFORMATIONS COMMODORE**, notre magazine "sans papier", est accessible par téléphone à l'aide d'un ordinateur et d'un modem Commodore.

En s'inscrivant à l'un de nos clubs d'ordinateur, on peut obtenir de l'aide pour résoudre un problème, parler à d'autres utilisateurs Commodore ou bénéficier de renseignements à jour sur les produits nouveaux, le logiciel et les ressources pédagogiques. L'utilisateur n'aura bientôt plus à taper les listes de programmes communiquées dans **POWER/PLAY** ou **COMMODORE**; il bénéficiera du téléchargement direct permis par le réseau d'informations (nouveau service prévu pour le début 1983). Avec ce service, l'utilisateur recevra la plupart des réponses à ses questions, presque avant de les avoir posées!

Pour communiquer avec notre magazine électronique, il faut disposer d'un modem et s'abonner à CompuServe™, un des plus grands réseaux de télécommunications des États-Unis. (Commodore facilite les choses en offrant un abonnement GRATUIT d'un an à CompuServe™ avec chaque VICMODEM.)

Il suffit d'appeler le numéro local de la banque de données CompuServe™ et de raccorder le téléphone au modem. Quand le texte vidéo de CompuServe™ apparaît sur l'écran, taper G CBM au clavier de l'ordinateur. Quand la table des matières ou "menu" du

RÉSEAU D'INFORMATIONS COMMODORE apparaît sur l'écran, choisir l'une des seize sections offertes et s'installer confortablement pour profiter du magazine "sans papier". Tous les autres magazines en parlent.

Pour plus de détails, on peut se rendre chez le détaillant Commodore ou s'adresser au service à la clientèle de CompuServe™ au 800-848-8990 (614-457-8600 pour l'État d'Ohio).

RÉSEAU D'INFORMATIONS COMMODORE

Description du menu principal	Détaillants Commodore
Codes d'accès direct	Ressources pédagogiques
Commandes spéciales	Groupes d'utilisateurs
Questions des utilisateurs	Descriptions
Tableau d'affichage public	Questions et réponses
Magazines et bulletins	Conseils sur le logiciel
Produits annoncés	Conseils techniques
Nouvelles Commodore en direct	Description des répertoires

CHAPITRE 1

RÈGLES DE PROGRAMMATION EN BASIC

- Introduction
- Codes d'affichage d'écran (jeu de caractères BASIC)
- Nombres et variables de programmation
- Expressions et opérateurs
- Techniques de programmation

INTRODUCTION

Le présent chapitre traite de la mémorisation et de la manipulation des données en BASIC. Il comprend:

1) Présentation brève des composants et fonctions du système d'exploitation et du jeu de caractères utilisé avec le Commodore 64.
2) Formation des constantes et des variables. Types de variables disponibles. Stockage des constantes et des variables en mémoire.
3) Règles des calculs arithmétiques, vérifications des relations, manipulation des chaînes et opérations logiques. Comprend également les règles de formation des expressions et les conversions des données nécessaires pour l'utilisation du BASIC avec des types de données mixtes.

CODES D'AFFICHAGE D'ÉCRAN (JEU DE CARACTÈRES BASIC)

SYSTÈME D'EXPLOITATION

Le système d'exploitation est logé dans les microplaquettes ou puces de mémoire morte ROM. Il se compose de trois modules de programmes distincts, mais reliés.

1) L'interpréteur BASIC
2) Le KERNAL
3) L'éditeur d'écran

1) **L'interpréteur BASIC** assure l'analyse de la syntaxe des instructions en BASIC et exécute les calculs requis et(ou) la manipulation des données. L'interpréteur BASIC possède un vocabulaire de 65 "mots clés" ayant des significations spéciales. L'alphabet en majuscules et en minuscules et les chiffres 0 à 9 servent à former les mots clés et les noms de variable. Certains caractères de ponctuation et symboles spéciaux ont aussi des sens particuliers à l'interpréteur. La table 1-1 indique les caractères spéciaux et leurs utilisations.
2) **Le KERNAL** assure la majeure partie du traitement au niveau des interruptions dans le système (pour plus de détails sur le traitement au niveau des interruptions, voir le chapitre 5). Le KERNAL assure aussi l'entrée et la sortie réelles des données.
3) **L'éditeur d'écran** commande la sortie vers l'écran vidéo (télécouleur) et la correction du texte de programme BASIC. En outre, l'éditeur d'écran intercepte les introductions au clavier pour décider s'il faut traiter immédiatement les caractères utilisés ou les transmettre à l'interpréteur BASIC.

Table 1-1. Jeu de caractères BASIC de CBM

CARACTÈRE	NOM et DESCRIPTION
	BLANC — sépare les mots clés et les noms de variable
;	POINT-VIRGULE — s'utilise dans les listes de variables pour mettre la sortie en forme
=	SIGNE D'ÉGALITÉ — attribution de valeurs et vérification de relations
+	SIGNE PLUS — addition arithmétique ou concaténation de chaînes (*concaténation:* réunion en une chaîne)
–	SIGNE MOINS — soustraction arithmétique ou signe moins unaire (–1)
*	ASTÉRISQUE — multiplication arithmétique
/	OBLIQUE — division arithmétique
↑	FLÈCHE MONTANTE — exposant arithmétique
(	PARENTHÈSE DE GAUCHE — évaluation d'une expression et fonctions
)	PARENTHÈSE DE DROITE — évaluation d'une expression et fonctions
%	POURCENTAGE — déclaration du nom de variable comme nombre entier
#	NUMÉRO — vient avant le numéro de fichier logique dans les instructions d'entrée et de sortie
$	SIGNE DOLLAR — déclare un nom de variable comme chaîne
,	VIRGULE — s'utilise dans les listes de variables pour mettre la sortie en forme; sépare aussi les paramètres de commande
.	POINT — point décimal dans les constantes à point flottant
"	GUILLEMET — délimite les constantes en chaîne
:	DEUX-POINTS — sépare les instructions BASIC multiples dans une ligne
?	POINT D'INTERROGATION — abréviation du mot clé PRINT
<	INFÉRIEUR À — s'utilise dans les vérifications de relations
>	SUPÉRIEUR À — s'utilise dans les vérifications de relations
π	PI — constante numérique 3.141592654

Le système d'exploitation permet deux modes d'utilisation du BASIC:

1) Le mode DIRECT
2) Le mode de PROGRAMME

1) Quand on utilise le mode DIRECT, les instructions en BASIC ne sont pas précédées de numéros de ligne. Elles sont exécutées quand on appuie sur la touche RETURN.
2) Le mode de PROGRAMME s'utilise pour l'exécution des programmes.

En mode de PROGRAMME, toutes les instructions BASIC doivent être accompagnées de numéros de ligne. On peut avoir plus d'une instruction BASIC sur une ligne de programme, mais le nombre d'instructions est limité parce qu'on ne peut introduire que 80 caractères dans une ligne logique d'écran. De ce fait, si l'on doit dépasser la limite de 80 caractères, on doit continuer l'instruction BASIC sur une nouvelle ligne, dotée d'un nouveau numéro.

REMARQUE: Toujours taper NEW et appuyer sur RETURN avant de commencer un nouveau programme.

Le Commodore 64 possède deux jeux complets de caractères que l'on peut utiliser dans les programmes ou à partir du clavier.

Dans le jeu 1, on dispose des majuscules et des chiffres 0 à 9 sans appuyer sur la touche SHIFT. Si l'on appuie sur la touche SHIFT pendant la frappe, on utilise les caractères graphiques du côté DROIT de l'avant des touches. Si l'on appuie sur la touche C= pendant la frappe, on utilise les caractères graphiques du côté GAUCHE de l'avant des touches. Si l'on appuie sur la touche SHIFT pendant la frappe d'un caractère sans symbole graphique à l'avant de la touche, on obtient le symbole indiqué à la partie supérieure de la touche.

Dans le jeu 2, on dispose de l'alphabet en minuscules et des chiffres 0 à 9 si l'on n'appuie pas sur la touche SHIFT. Si l'on appuie sur la touche SHIFT pendant la frappe, on obtient l'alphabet en majuscules. Cette fois encore, on affiche les symboles graphiques de la GAUCHE de l'avant des touches si l'on appuie sur la touche C=. Si l'on appuie sur la touche SHIFT pendant la frappe d'une touche sans caractère graphique, on obtient le symbole de la partie supérieure de la touche.

Pour passer d'un jeu de caractères à l'autre, appuyer simultanément sur les touches C= et SHIFT.

NOMBRES ET VARIABLES DE PROGRAMMATION

CONSTANTES ENTIÈRES, À POINT FLOTTANT ET EN CHAÎNE

Les *constantes* sont des valeurs de données que l'on place dans les instructions en BASIC. Le BASIC utilise ces valeurs pour présenter les données pendant l'exécution des instructions. Le BASIC CBM peut distinguer et manipuler trois types de constantes:

1) NOMBRES ENTIERS
2) NOMBRES À POINT FLOTTANT
3) CHAÎNES

Les **constantes entières** sont des nombres entiers (nombres sans point décimal). Les constantes entières doivent être comprises entre −32768 et +32767. *Les constantes entières n'ont ni points décimaux ni virgules entre les chiffres.* Si le signe plus (+) est omis, on suppose que la constante est positive. Les zéros qui précèdent une constante sont ignorés; ne pas les utiliser, car ils gaspillent de la place de mémoire et ralentissent le programme. Ils ne peuvent cependant pas causer d'erreurs. Les constantes entières sont mémorisées sous forme de nombres binaires à deux octets. Voici quelques exemples de constantes entières:

- 12
8765
- 32768
+ 44
0
- 32767

REMARQUE: NE mettre AUCUNE virgule dans un nombre. Par exemple, toujours taper 32,000 sous la forme 32000. Si l'on introduit une virgule dans un nombre, on obtient le message d'erreur BASIC **?SYNTAX ERROR (Erreur de syntaxe).**

Les **constantes à point flottant** sont des nombres positifs et négatifs qui peuvent contenir des fractions. On peut indiquer les parties fractionnaires d'un nombre avec un point décimal. Cette fois encore, NE PAS utiliser de virgule entre les nombres. Si l'on omet le signe plus (+) devant un nombre, le Commodore 64 suppose que celui-ci est positif. Si l'on ne met pas le point décimal, l'ordinateur suppose qu'il suit le dernier chiffre du nombre. Comme pour les nombres entiers, les zéros qui précèdent une constante sont ignorés. On peut utiliser les constantes à point flottant de deux façons:

1) NOMBRE SIMPLE
2) NOTATION SCIENTIFIQUE

Les constantes à point flottant apparaissent sur l'écran avec un maximum de neuf chiffres. Ces chiffres peuvent représenter des valeurs comprises entre −999999999. et +999999999. Si l'on introduit plus de neuf chiffres, le nombre est arrondi en fonction du dixième chiffre. Si le dixième chiffre est supérieur ou égal à 5, le nombre est arrondi à l'unité supérieure. Si ce chiffre est inférieur à 5, le nombre est arrondi à l'unité inférieure. Cette indication peut influer sur les totaux finals quand on travaille avec certains nombres.

Les nombres à points flottants sont mémorisés (avec cinq octets de mémoire) et manipulés dans les calculs avec une précision de dix positions. À l'impression des résultats, les nombres sont cependant arrondis à neuf chiffres. Voici quelques exemples de nombres simples à point flottant:

1.23
-.998877
+3.1459
.7777777
-333.
.01

Les nombres inférieurs à .01 ou supérieurs à 999999999. sont imprimés en *notation scientifique*. Dans cette notation, une constante à point flottant se compose de trois parties:

1) LA MANTISSE
2) LA LETTRE E
3) L'EXPOSANT

La *mantisse* est un nombre simple à point flottant. La lettre E indique que le nombre est présenté sous forme exponentielle. E représente donc *10 (3E3=3*10↑3=3000). L'exposant correspond à la puissance de 10 à laquelle on doit élever le nombre.

La mantisse et l'exposant sont accompagnés du signe + ou −. L'intervalle de l'exposant varie de −39 à +38; il indique le nombre de positions de déplacement vers la gauche (−) ou vers la droite (+) du point décimal réel de la mantisse si la valeur de la constante était présentée sous la forme d'un nombre simple.

Il existe une limite à la grandeur des nombres à point flottant que le BASIC peut manipuler, même en notation scientifique. Le nombre le plus grand est +1.70141183E+38; les calculs qui conduisent à un nombre plus grand se traduisent par le message d'erreur BASIC **?OVERFLOW ERROR** (dépassement de capacité). Le nombre à point flottant le plus petit est +2.93873588E−39; les calculs qui conduisent à une valeur plus petite donnent une réponse nulle (zéro) sans message d'erreur. Nous donnons ci-dessous quelques exemples de nombres à point flottant en notation scientifique (avec leurs valeurs décimales):

235.988E-3	(.235988)
2359E6	(2359000000.)
-7.09E-12	(-.00000000000709)
-3.14159E+5	(-314159.)

Les **constantes en chaîne** sont des groupes d'informations alphanumériques tels que des lettres, des nombres et des symboles. Quand on introduit une chaîne à partir du cla-

vier, elle peut avoir n'importe quelle longueur jusqu'à concurrence de l'espace disponible dans une ligne de 80 caractères (c'est-à-dire les espaces de caractères qui NE sont PAS occupés par le numéro de ligne et les autres parties requises de l'instruction).

Une constante en chaîne peut contenir des blancs, des lettres, des nombres, des signes de ponctuation et des caractères couleur ou de commande de curseur, en toute combinaison. On peut même placer des virgules entre les chiffres. *Le seul caractère interdit dans une chaîne est le guillemet ("),* car il sert à définir le début et la fin de la chaîne. Une chaîne peut aussi avoir une valeur nulle; elle ne contient alors aucune donnée de caractère. On peut omettre le guillemet de fin d'une chaîne s'il constitue le dernier élément d'une ligne ou s'il est suivi d'un deux-points (:). Voici quelques exemples de constantes en chaîne:

"" (chaîne nulle)
"BONJOUR"
"$25,000.00"
"NOMBRE D'EMPLOYÉS"

REMARQUE: Utiliser CHR$(34) pour inclure des guillemets (") dans une chaîne.

VARIABLES ENTIÈRES, À POINT FLOTTANT ET EN CHAÎNE

Les *variables* correspondent à des noms qui représentent des valeurs de données utilisées dans les instructions en BASIC. On peut attribuer la valeur représentée par une variable en la fixant égale à une constante ou au résultat de calculs dans un programme. Les données de variable, comme les constantes, peuvent être des nombres entiers, des nombres à point flottant ou des chaînes. Si l'on mentionne un nom de variable dans un programme avant qu'une valeur lui ait été attribuée, l'interpréteur BASIC crée automatiquement la variable avec une valeur nulle s'il s'agit d'un nombre entier ou à point flottant. Il peut aussi créer une variable de valeur nulle si on utilise des chaînes.

La longueur des noms de variable est sans importance, mais les deux premiers caractères sont seuls pris en considération en BASIC CBM. De ce fait, les deux premiers caractères des noms utilisés pour les variables doivent être différents. *Les noms de variable NE doivent PAS correspondre à des mots clés de BASIC; ils NE peuvent PAS non plus contenir de mots clés en leur milieu.* Les mots clés comprennent les commandes, les instructions, les noms de fonction et les noms d'opérateur logique de BASIC. Si l'on utilise par mégarde un mot clé au milieu d'un nom de variable, le message d'erreur BASIC **?SYNTAX ERROR** (erreur de syntaxe) apparaît sur l'écran.

Pour la formation des noms de variable, on utilise les caractères de l'alphabet et les chiffres 0 à 9. Le premier caractère du nom doit être une lettre. Pour le deuxième caractère

du nom, on peut utiliser les caractères de déclaration de données (%) et ($). Le signe de pourcentage (%) déclare la variable comme nombre entier et le signe dollar ($) déclare une variable en chaîne. Si l'on n'utilise aucun caractère de déclaration, l'interpréteur suppose une variable à point flottant. Nous donnons ci-après quelques exemples de noms de variable, d'attributions de valeur et de types de données.

A$ = "VENTES BRUTES"	(variable en chaîne)
MTH$ = "JANV" + A$	(variable en chaîne)
K% = 5	(variable entière)
CNT% = CNT% + 1	(variable entière)
FP = 12.5	(variable à point flottant)
SOMME = FP*CNT%	(variable à point flottant)

TABLEAUX ENTIERS, À POINT FLOTTANT ET EN CHAÎNE

Un *tableau* est une liste ou table d'éléments de données associés qui correspondent à un seul nom de variable. Un tableau est ainsi une suite de variables connexes. Par exemple, on peut considérer une table de nombres comme un tableau. Les nombres individuels de la table forment les "éléments" du tableau.

Les tableaux permettent de représenter sous forme abrégée un grand nombre de variables apparentées. Prenons par exemple une table de nombres et supposons qu'elle comprenne 10 rangées de 20 nombres chacune, soit un total de 200 nombres. Si l'on ne disposait pas d'un simple nom de tableau, on devrait attribuer un nom particulier à chaque valeur de la table. Avec l'utilisation des tableaux, on n'a cependant besoin que d'un seul nom; tous les éléments du tableau sont identifiés par leur position particulière dans le tableau.

Les noms de tableau peuvent être représentés par des données entières, des données à point flottant ou des données en chaîne. Tous les éléments d'un tableau ont le même type de données que le nom du tableau. Les tableaux peuvent avoir une simple dimension (liste simple, par exemple), des dimensions multiples (par exemple, une grille représentée par des rangées et des colonnes ou un cube de Rubik®). Chaque élément d'un tableau possède une identification unique représentée par un indice inférieur (ou variable d'index) qui suit le nom du tableau; l'indice inférieur est indiqué entre parenthèses ( ).

En théorie, un tableau peut avoir un nombre maximal de 255 dimensions; dans chaque dimension, le nombre d'éléments est limité à 32767. Pour des raisons pratiques, les dimensions des tableaux sont souvent limitées par la place de mémoire disponible pour recevoir leurs données et(ou) par la ligne d'écran logique de 80 caractères. Si un tableau n'a qu'une dimension et si sa valeur d'indice inférieur ne dépasse jamais 10 (11 éléments de 0 à 10), il est alors créé par l'interpréteur et rempli de zéros (ou d'espaces nuls pour le type en chaîne) la première fois qu'il est fait référence à un élément du tableau; sinon, on

doit utiliser l'instruction DIM en BASIC pour définir la forme et la taille du tableau. La quantité de mémoire nécessaire au stockage d'un tableau se détermine ainsi:

	5	octets pour le nom du tableau
	+ 2	octets pour chaque dimension du tableau
	+ 2	octets par élément pour les données entières
OU	+ 5	octets par élément pour les données à point flottant
OU	+ 3	octets par élément pour les données en chaîne
ET	+ 1	octet par caractère dans chaque élément de chaîne

On peut représenter les indices inférieurs par des constantes entières, des variables ou une expression arithmétique qui donne un résultat entier. Les indices inférieurs séparés par des virgules sont requis pour chaque dimension d'un tableau. Les indices inférieurs peuvent avoir des valeurs variant de 0 jusqu'au nombre d'éléments des dimensions respectives du tableau. Les valeurs en dehors de cet intervalle amènent le message d'erreur BASIC **?BAD SUBSCRIPT** (indice inférieur erroné). Voici quelques exemples de noms de tableau, d'attributions de valeur et de types de données:

A$(0) = "VENTES BRUTES"	(tableau de données en chaîne)
MTH$(K%) = "JANV"	(tableau de données en chaîne)
G2%(X) = 5	(tableau de données entières)
CNT%(G2%(X)) = CNT%(1) − 2	(tableau de données entières)
FP(12*K%) = 24.8	(tableau de données à point flottant)
SOMME(CNT%(1)) = FP↑K%	(tableau de données à point flottant)

A(5) = 0	(donne la valeur 0 au 5e élément du tableau à une dimension nommé "A")
B(5,6) = 0	(donne la valeur 0 à la rangée de position 5 et à la colonne de position 6 dans le tableau à 2 dimensions nommé "B")
C(1,2,3) = 0	(donne la valeur 0 à l'élément de la rangée de position 1, à la colonne de position 2 et à la profondeur de position 3 dans le tableau à 3 dimensions nommé "C")

EXPRESSIONS ET OPÉRATEURS

Les expressions se forment avec des constantes, des variables et(ou) des tableaux. Une expression peut se composer d'une constante simple, d'une variable simple ou d'une variable en tableau de tout type. Elle peut aussi se constituer de constantes et de variables avec des opérateurs arithmétiques, logiques ou de relation prévus pour don-

ner une seule valeur. Nous expliquerons ci-dessus le rôle des opérateurs. On peut répartir les expressions en deux catégories:

1) ARITHMÉTIQUE
2) CHAÎNE

Les expressions se composent normalement de deux ou plusieurs éléments de données dits facteurs ou *opérandes*. Les facteurs sont séparés par un opérateur pour obtenir le résultat désiré. Dans ce but, on attribue généralement la valeur de l'expression à un nom de variable. Tous les exemples de constantes et de variables que nous avons vus jusqu'à présent représentaient aussi des expressions.

Un opérateur est un symbole spécial. Pour l'interpréteur BASIC du Commodore 64, un opérateur indique une opération à exécuter sur les variables ou les données constantes. La combinaison d'un ou plusieurs opérateurs avec une ou plusieurs variables et(ou) constantes forme une expression. Le BASIC du Commodore 64 distingue les opérateurs arithmétiques, logiques et de relation.

EXPRESSIONS ARITHMÉTIQUES

Les expressions arithmétiques résolues donnent une valeur entière ou à point flottant. Les opérateurs arithmétiques (+ , − , *,/, ↑) servent à exécuter respectivement l'addition, la soustraction, la multiplication, la division et l'élévation à une puissance.

OPÉRATIONS ARITHMÉTIQUES

Un opérateur arithmétique définit une opération exécutée sur les deux facteurs qu'il sépare. Les opérations arithmétiques se font avec des nombres à point flottant. Les nombres entiers sont convertis en nombres à point flottant avant l'exécution d'une opération arithmétique. Le résultat est reconverti en nombre entier s'il est attribué à un nom de variable entière.

ADDITION (+): Le signe plus (+) précise que le facteur de droite est ajouté au facteur de gauche.

EXEMPLES:

```
2 + 2
A + B + C
X% + 1
BR + 10E - 2
```

SOUSTRACTION (–): Le signe moins (–) précise que le facteur de droite est soustrait du facteur de gauche.

EXEMPLES:

4 – 1
100 – 64
A – B
55 – 142

Le signe moins peut aussi servir de moins unaire; dans ce cas, il précède un nombre négatif. Cette disposition équivaut à soustraire le nombre de zéro.

EXEMPLES:

–5
–9E4
–B
4– (–2) équivaut à 4+2

MULTIPLICATION (*): Un astérisque (*) précise que le facteur de gauche est multiplié par le facteur de droite.

EXEMPLES:

100*2
50*0
A*X1
R%*14

DIVISION (/): L'oblique (/) précise que le facteur de gauche est divisé par le facteur de droite.

EXEMPLES:

10/2
6400/4
A/B
4E2/XR

ÉLÉVATION À UNE PUISSANCE (↑): La flèche montante (↑) précise que le facteur de gauche est élevé à la puissance précisée par le facteur de droite (exposant). Si le facteur

de droite est égal à 2, le nombre de gauche est élevé au carré; si l'exposant est 3, le nombre est élevé au cube, etc. L'exposant peut correspondre à tout nombre pourvu que le résultat de l'opération donne un nombre valable à point flottant.

EXEMPLES:

2↑2	Équivaut à: 2*2
3↑3	Équivaut à: 3*3*3
4↑4	Équivaut à: 4*4*4*4
AB↑CD	
3↑ – 2	Équivaut à: 1/3*1/3

OPÉRATEURS DE RELATION

Les opérateurs de relation (< , =, >, < =, > =, < >) servent essentiellement à comparer les valeurs de deux facteurs, mais ils donnent aussi un résultat arithmétique. Utilisés dans des comparaisons, les opérateurs de relation et les opérateurs logiques (ET, OU et NON) donnent une évaluation arithmétique vraie/fausse d'une expression. Si la relation indiquée dans l'expression est vraie, le résultat reçoit une valeur entière de –1; si la relation est fausse, le résultat reçoit la valeur 0. Les opérateurs de relation sont:

<	INFÉRIEUR À
=	ÉGAL À
>	SUPÉRIEUR À
< =	INFÉRIEUR OU ÉGAL À
> =	SUPÉRIEUR OU ÉGAL À
< >	N'EST PAS ÉGAL À

EXEMPLES:

1 = 5 – 4	résultat vrai (– 1)
14 > 66	résultat faux (0)
15 > = 15	résultat vrai (– 1)

On peut utiliser les opérateurs de relation pour comparer les chaînes. Dans les comparaisons, les lettres de l'alphabet sont considérées dans l'ordre A < B < C < D, etc. On compare les chaînes en évaluant la relation entre les caractères correspondants de gauche à droite (voir "Opérations sur les chaînes").

EXEMPLES:

"A" < "B"	résultat vrai (– 1)
"X" = "YY"	résultat faux (0)
BB$ < > CC$	

On ne peut comparer ou attribuer les éléments de données numériques qu'à d'autres éléments numériques. Il en est de même pour la comparaison des chaînes, sinon on obtient le message d'erreur BASIC **?TYPE MISMATCH** (non-concordance de types). Pour comparer les facteurs numériques, on convertit d'abord les valeurs de chacun d'eux (ou des deux) de la forme entière à la forme à point flottant, suivant le cas. On évalue ensuite la relation des valeurs à point flottant pour arriver à un résultat vrai/faux.

À la fin de toutes les comparaisons, on obtient une valeur entière, quel que soit le type de données du facteur (même si les deux facteurs sont des chaînes). De ce fait, on peut utiliser une comparaison de deux facteurs comme facteur dans l'exécution des calculs. Le résultat (−1 ou 0) peut servir de facteur quelconque, mais pas de diviseur, car la division par 0 est impossible.

OPÉRATEURS LOGIQUES

On peut utiliser les opérateurs logiques (ET, OU et NON) pour modifier le sens des opérateurs de relation ou pour arriver à un résultat arithmétique. Les opérateurs logiques peuvent donner des résultats autres que −1 et 0. Tout résultat non nul est cependant considéré vrai dans la vérification d'un état vrai/faux.

On peut aussi utiliser les opérateurs logiques ou opérateurs *booléens* pour exécuter des opérations logiques avec des chiffres binaires (bits) individuels dans deux facteurs. Toutefois, quand on utilise l'opérateur NON, l'opération ne se fait que sur le seul facteur de droite. Les facteurs doivent être compris dans l'intervalle de valeurs entières de −32768 à +32767 (les nombres à point flottant sont convertis en valeurs entières). Les opérations logiques donnent un résultat entier.

Les opérations logiques se font de bit à bit correspondant sur les deux facteurs. Le ET logique donne un résultat binaire de 1, uniquement si les deux bits des facteurs sont aussi 1. Le OU logique donne un résultat binaire de 1 si l'un ou l'autre des bits de facteur vaut 1. Le NON logique correspond à la valeur opposée de chaque bit pris comme facteur simple. Il revient en fait à dire "si la valeur N'EST PAS 1, elle est donc de 0; si elle N'EST PAS de zéro, elle est donc de 1".

Le OU exclusif n'a pas d'opérateur logique; il s'exécute dans l'instruction WAIT. Avec le OU exclusif, si les bits de deux facteurs sont égaux, le résultat est 0; sinon, il vaut 1.

Les opérations logiques se définissent par des groupes d'instructions qui, combinés, donnent une "table de vérité" de Boole (voir table 1-2).

Table 1-2. Table de vérité booléenne

L'opération ET ne donne 1 que si les deux bits sont 1:

1 ET 1 = 1
0 ET 1 = 0
1 ET 0 = 0
0 ET 0 = 0

L'opération OU donne 1 si l'un des bits est 1:

1 OU 1 = 1
0 OU 1 = 1
1 OU 0 = 1
0 OU 0 = 0

L'opération NON est le complément logique de chaque bit:

NON 1 = 0
NON 0 = 1

Le OU exclusif (XOU) fait partie de l'instruction WAIT:

1 XOU 1 = 0
1 XOU 0 = 1
0 XOU 1 = 1
0 XOU 0 = 0

Les opérateurs logiques ET, OU et NON précisent une expression arithmétique booléenne à exécuter sur les deux expressions de facteur de chaque côté de l'opérateur. Dans les cas de NON, SEUL le facteur de DROITE est envisagé. Les opérations logiques en arithmétique booléenne ne sont exécutées que si les opérations arithmétiques et de relation d'une expression ont été calculées.

EXEMPLES:

IF A = 100 AND B = 100 THEN 10	(si A et B ont tous deux la valeur 100, le résultat est donc vrai)
A = 96 AND 32: PRINT A	(A = 32)

IF A = 100 OR B = 100 THEN 20	(si A ou B vaut 100, le résultat est donc vrai)
A = 64 OR 32: PRINT A	(A = 96)
IF NOT X < Y THEN 30	(si X> = Y, le résultat est vrai)
X = NOT 96	(le résultat est —97 (complément à deux))

HIÉRARCHIE DES OPÉRATIONS

Les expressions exécutent les différentes opérations suivant une hiérarchie établie. Certaines opérations sont ainsi exécutées avant d'autres. On peut modifier l'ordre normal des opérations en plaçant deux ou plusieurs facteurs entre parenthèses ( ) afin de créer une "expression secondaire". Les parties d'une expression entre parenthèses sont réduites à une seule valeur avant de passer aux parties en dehors des parenthèses.

Quand on utilise des parenthèses dans des expressions, on doit les utiliser en paires, de façon à toujours avoir le même nombre de parenthèses gauches et de parenthèses droites. Si ce n'est pas le cas. le message d'erreur BASIC **?SYNTAX ERROR** (erreur de syntaxe) apparaît.

On peut placer entre parenthèses des expressions ayant déjà des facteurs entre parenthèses afin de former des expressions complexes à plusieurs niveaux. Cette opération correspond à *l'emboîtement*. On peut emboîter des parenthèses dans des expressions jusqu'à un maximum de dix niveaux, soit dix jeux correspondants de parenthèses. Les calculs commencent avec les expressions les plus intérieures. Voici quelques exemples d'expressions:

```
A+B
C \ (D+E)/2
((X—C \ (D+E)/2)*10)+1
GG$>HH$
JJ$+"MORE"
K%=1 AND M < >X
K%=2 OR (A=B AND M<X)
NOT (D=E)
```

L'interpréteur BASIC exécute normalement les opérations des expressions en calculant d'abord les opérations arithmétiques avant de passer aux opérations de relation et de terminer par les opérations logiques. Les opérateurs arithmétiques et logiques suivent un

ordre de priorité (ou hiérarchie des opérations) entre eux. Par ailleurs, les opérateurs de relation n'ont pas d'ordre de priorité; l'expression est évaluée de la gauche vers la droite.

Si tous les opérateurs restants d'une expression ont le même niveau de priorité, les opérations sont alors exécutées de la gauche vers la droite. Pendant l'exécution des expressions entre parenthèses, l'ordre normal de priorité est respecté. La table 1-3 montre la hiérarchie des opérations arithmétiques et logiques, dans l'ordre de priorité du début à la fin.

Table 1-3. Hiérarchie des opérations exécutées dans les expressions

OPÉRATEUR	DESCRIPTION	EXEMPLE
↑	Élévation à une puissance	BASE ↑ EXP
—	Négation (moins unaire)	— A
* —	Multiplication Division	AB * CD EF / GH
+ —	Addition Soustraction	CNT + 2 JK — PQ
< = >	Opérations de relation	A< = B
NOT	NON logique (Complément entier à deux)	NOT K%
AND	ET logique	JK AND 128
OR	OU logique	PQ OR 15

OPÉRATIONS SUR LES CHAÎNES

On compare les chaînes avec les mêmes opérateurs de relation (=, < >, < =, > =, <, >) que pour des nombres. Dans les comparaisons des chaînes, on considère un caractère à la fois (de la gauche vers la droite) dans chaque chaîne et l'on évalue chaque position de code de caractère du jeu de caractères PET/CBM. Si des codes de caractères sont identiques, les caractères sont égaux. Si les codes diffèrent, le caractère au numéro de code inférieur est plus bas dans le jeu de caractères. La comparaison cesse quand on

arrive à la fin de chaque chaîne. Toutes choses étant égales, la chaîne la plus courte est considérée inférieure à la chaîne la plus longue. Les *blancs à gauche ou à droite sont significatifs.*

À la fin de toutes les comparaisons, on obtient un résultat entier, quels que soient les types de données. Cette remarque est vraie, même si les deux facteurs sont des chaînes. De ce fait, on peut utiliser une comparaison de deux facteur en chaîne comme facteur dans l'exécution de calculs. Le résultat obtenu est −1 ou 0 (vrai ou faux); on peut l'utiliser comme facteur, excepté comme diviseur, car la division par 0 est impossible.

EXPRESSIONS EN CHAÎNE

Les expressions sont considérées comme si un "< >0" implicite les suit. Si une expression est vraie, les instructions BASIC suivantes dans la même ligne de programme sont alors exécutées. Si l'expression est fausse, le reste de la ligne est ignoré et la ligne suivante du programme est exécutée.

Comme pour les nombres, on peut aussi exécuter des opérations avec les variables en chaîne. Le seul opérateur arithmétique en chaîne accepté par le BASIC CBM est le signe plus (+) qui sert à la concaténation des chaînes. Dans la concaténation, la chaîne à droite du signe plus est annexée à la chaîne à gauche; il en résulte une troisième chaîne. On peut imprimer immédiatement le résultat, l'utiliser dans une comparaison ou l'attribuer à un nom de variable. Si l'on compare un élément de données en chaîne ou si on le rend égal à un élément numérique (ou vice versa), on obtient le message d'erreur BASIC **?TYPE MISMATCH** (non-concordance de type). Nous donnons ci-dessous quelques exemples d'expressions en chaîne et de concaténation:

```
10 A$ = "NOM DE" : B$ = "FICHIER"
20 NOM$ = A$ + B$                  (donne la chaîne: NOM DE FICHIER)
30 RES$ = "NOUVEAU " + A$ + B$     (donne la chaîne: NOUVEAU NOM DE
                                    FICHIER)
                ↑
                └──── Noter l'espace.
```

TECHNIQUES DE PROGRAMMATION

CONVERSIONS DES DONNÉES

Le cas échéant, l'interpréteur BASIC CBM convertit un élément de données numérique d'une valeur entière en une valeur à point flottant ou vice versa en appliquant les règles suivantes:

- Toutes les opérations arithmétiques et de relation sont exécutées en format à point flottant. Les valeurs entières sont converties sous forme à point flottant pour l'évaluation de l'expression. Le résultat est reconverti en valeur entière. Les opérations logiques convertissent leurs facteurs en valeurs entières et donnent un résultat entier.
- Si un nom de variable numérique d'un type est fixé équivalent à un élément de donnée numérique de type différent, le nombre est converti et mémorisé suivant le type de données déclaré dans le nom de variable.
- Quand une valeur à point flottant est convertie en valeur entière, la partie fractionnaire est tronquée (éliminée) et le résultat entier est inférieur ou égal à la valeur à point flottant. Si le résultat est en dehors de l'intervalle de + 32767 à − 32768, le message d'erreur BASIC **?ILLEGAL QUANTITY** (quantité non conforme) apparaît.

UTILISATION DE L'INSTRUCTION INPUT

Nous connaissons maintenant la nature des variables. Réunissons donc nos connaissances avec l'instruction INPUT et passons à quelques applications pratiques de programmation.

Dans notre premier exemple, on peut imaginer une variable comme un "compartiment de mémoire" où le Commodore 64 remise la réponse de l'utilisateur à la question du message-guide. Pour écrire un programme qui demande à l'utilisateur de taper un nom, on peut attribuer la variable N$ au nom tapé. Chaque fois que PRINT N$ apparaît dans le programme, le Commodore 64 imprime automatiquement le nom tapé par l'utilisateur.

Taper le mot NEW sur le Commodore 64 et appuyer sur la touche **RETURN**. Essayer l'exemple suivant:

```
10 PRINT "NOM DE L'UTILISATEUR":INPUT N$
20 PRINT "BONJOUR," N$
```

Dans cet exemple, on a utilisé N pour se rappeler que cette variable correspond à "NOM". Le signe dollar ($) indique à l'ordinateur que l'on utilise une variable en chaîne. On doit absolument différencier les deux types suivants de variable:

1) NUMÉRIQUE
2) CHAÎNE

On se souvient probablement avoir vu, dans les sections précédentes, que les variables numériques servent à mémoriser des valeurs numériques comme 1, 100, 4000, etc. Une variable numérique peut être représentée par une seule lettre (A), par deux lettres (AB), par une lettre et un chiffre (A1) ou par deux lettres et un chiffre (AB1). On peut économiser de la place de mémoire en utilisant des variables plus courtes. On peut aussi utiliser des lettres et des chiffres pour différentes catégories dans le même programme (A1, A2, A3). Si l'on désire une réponse avec des nombres entiers au lieu de nombres avec une partie décimale, il suffit d'insérer un signe de pourcentage (%) à la fin du nom de variable (AB%, A1%, etc.).

Étudions maintenant quelques exemples où l'on utilise différents types de variables et d'expressions avec l'intsruction INPUT.

```
10 PRINT "ENTRER UN NOMBRE":INPUT A
20 PRINT A
```

```
10 PRINT "ENTRER UN MOT":INPUT A$
20 PRINT A$
```

```
10 PRINT "ENTRER UN NOMBRE":INPUT A
20 PRINT A "PAR 5 EGALE" A*5
```

REMARQUE: L'exemple 3 montre que les messages ou les messages-guides sont placés entre les guillemets (" "), alors que les variables sont à l'extérieur. Il faut aussi remarquer que, à la ligne 20, la variable A a été imprimée en premier, suivie du message "PAR 5 EGALE", puis du calcul de multiplication de la variable A par 5 (A*5).

Les calculs jouent un grand rôle dans la plupart des programmes. On peut utiliser des "nombres réels" ou des variables, mais avec les nombres fournis par un utilisateur, on doit utiliser des variables numériques. Pour commencer, demander à l'utilisateur de taper deux nombres:

```
10 PRINT "TAPER 2 NOMBRES":INPUT A:INPUT B
```

EXEMPLE DE BUDGET DE REVENU/DEPENSES

```
5  PRINT "♥" — SHIFT CLR/HOME
10  PRINT"REVENU MENSUEL":INPUT RE
20  PRINT
30  PRINT"CATEGORIE DE DEPENSES 1":INPUT D1$
40  PRINT"MONTANT DEPENSE":INPUT D1
50  PRINT
60  PRINT"CATEGORIE DE DEPENSES 2":INPUT D2$
70  PRINT"MONTANT DEPENSE":INPUT D2
80  PRINT
90  PRINT"CATEGORIE DE DEPENSES 3":INPUT D3$
100  PRINT"MONTANT DEPENSE":INPUT D3
110  PRINT "♥" — SHIFT CLR/HOME
120 D = D1 + D2 + D3
130 PD = D/RE
140  PRINT"REVENU MENSUEL:     $"RE
150  PRINT"DEPENSES TOTALES:     $"D
160  PRINT"SOLDE EGAL A:   $"RE—D
170  PRINT
180  PRINT D1$" = "(D1/D)*100"% DE DEPENSES TOTALES"
190  PRINT D2$"="(D2/D)*100"% DE DEPENSES TOTALES"
200  PRINT D3$"="(D3/D)*100"% DE DEPENSES TOTALES"
210  PRINT
220  PRINT "VOS DEPENSES = "PD*100"% DE VOTRE REVENU TOTAL"
230 FOR X = 1TO5000:NEXT:PRINT
240  PRINT"REPETER? (OUI OU NON)":INPUT OUI$:IF OUI$ = "OUI"THEN5
250  PRINT "♥" :END
          SHIFT CLR/HOME
```

REMARQUE: RE NE PEUT PAS être égal à 0; D1, D2 et D3 NE PEUVENT PAS être toutes égales à zéro en même temps.

EXPLICATION DE TAILLE DE L'EXEMPLE DE BUDGET DE REVENU/DÉPENSES

Ligne(s)	Description
5	Efface l'écran.
10	Instruction PRINT/INPUT.
20	Insère une ligne vide.
30	Catégorie de dépenses 1 = D1$.
40	Montant dépensé = D1.
50	Insère une ligne vide.
60	Catégorie de dépenses 2 = D2$.
70	Montant dépensé 2 = D2.
80	Insère une ligne vide.
90	Catégorie de dépenses 3 = D3$.
100	Montant dépensé 3 = D3.
110	Efface l'écran.
120	Ajoute les montants dépensés = D.
130	Calcule le pourcentage dépenses/revenu.
140	Affiche le revenu.
150	Affiche les dépenses totales.
160	Affiche revenu − dépenses.
170	Insère une ligne vide.
180-200	Les lignes 180 à 200 calculent le pourcentage de chaque montant dépensé par rapport aux dépenses totales.
210	Insère une ligne vide.
220	Affiche % dépenses/revenu.
230	Boucle de retard.

Multiplier maintenant ces deux nombres pour obtenir une nouvelle variable C comme l'indique la ligne 20 ci-dessous:

```
20 C = A*B
```

Pour imprimer le résultat sous forme de message, taper:

```
30 PRINT A "FOIS" B "ÉGALE" C
```

Introduire ces 3 lignes et exécuter (RUN) le programme. Remarquer que les messages sont placés entre les guillemets et que les variables ne le sont pas.

Supposons maintenant que l'on désire un signe dollar ($) devant le nombre représenté par la variable C. Le $ doit être imprimé entre guillemets et devant la variable C. Pour ajouter le $ au programme, appuyer sur les touches **RUN/STOP** et **RESTORE**. Taper ensuite la ligne 40 suivante:

```
40 PRINT "$" C
```

Appuyer sur **RETURN**, taper RUN et appuyer de nouveau sur **RETURN**.

Le signe dollar se place entre guillemets, car la variable C ne représente qu'un nombre et ne peut pas contenir un signe $. Si le nombre représenté par C était 100, l'écran du Commodore 64 indiquerait alors $ 100. Si l'on essaie toutefois PRINT $C sans utiliser de guillemets, on obtient un message **?SYNTAX ERROR** (erreur de syntaxe).

Conseil sur le $$$: On peut créer une variable représentant un signe dollar que l'on peut ensuite utiliser à la place du $ quand on veut l'utiliser dans des variables numériques. Par exemple:

```
10 Z$ = "$"
```

Quand on désire un signe dollar, on peut maintenant utiliser la variable en chaîne Z$. Essayer ce programme:

```
10 Z$ = "$":INPUT A
20 PRINT Z$A
```

La ligne 10 définit $ comme variable en chaîne Z$ et entre (INPUT) ensuite un nombre A. La ligne 20 imprime Z$ ($) à côté de A (nombre).

On s'apercevra probablement qu'il est plus facile d'attribuer certains caractères, comme le signe dollar, à une variable en chaîne que de taper "$" chaque fois que l'on désire calculer des sommes en dollars ou d'autres éléments faisant appel aux " " comme le %.

UTILISATION DE L'INSTRUCTION GET

Dans les programmes les plus simples, on utilise l'instruction INPUT pour obtenir les données de l'utilisateur de l'ordinateur. Dans le cas de besoins plus complexes, comme la protection contre les erreurs de frappe, l'instruction GET offre davantage de flexibilité et donne un programme plus "intelligent". La présente section montre l'utilisation de l'instruction GET pour ajouter des caractéristiques spéciales de correction d'écran aux programmes.

Le Commodore 64 possède une mémoire tampon de clavier qui contient jusqu'à 10 caractères. Si l'ordinateur est occupé à une autre tâche et s'il ne lit pas le clavier, on peut quand même taper jusqu'à 10 caractères qui seront lus dès que le Commodore 64 aura terminé sa tâche présente. Pour le montrer, taper le programme suivant sur le Commodore 64:

```
NEW
10 TI$ = ''000000''
20 IF TI$ < ''000015'' THEN 20
```

Exécuter (RUN) le programme et appuyer sur **RETURN** . Pendant l'exécution du programme, taper BONJOUR.

Rien ne se passe pendant 15 secondes environ après le début du programme. Le message BONJOUR n'apparaît sur l'écran qu'après un certain temps.

Supposons que l'on fasse la queue au guichet d'un cinéma. La première personne de la queue est la première à recevoir son billet et à quitter la queue. La dernière personne reçoit son billet en dernier. L'instruction GET joue le rôle de distributeur de billets. Elle détermine d'abord s'il y a une queue de caractères, c'est-à-dire si l'on a appuyé sur des touches. Si la réponse est affirmative, les caractères sont alors placés dans les variables appropriées. Si on n'a appuyé sur aucune touche, une valeur nulle est alors attribuée à une variable.

À ce point, il faut remarquer que si l'on essaie de placer plus de 10 caractères à la fois dans la mémoire tampon, ceux qui suivent le 10e sont perdus.

L'instruction GET continue sa tâche, même si l'on n'a tapé aucun caractère; de ce fait, il faut souvent la placer dans une boucle, de façon qu'elle attende la frappe d'une touche ou la réception d'un caractère du programme.

Nous donnons ci-dessous la forme recommandée de l'instruction GET. Taper NEW pour effacer le programme précédent.

```
10 GET A$ : IF A$ = ''''THEN 10
```

On peut remarquer qu'il n'y a pas d'espace entre les guillemets ('''') sur cette ligne. On indique ainsi une valeur nulle et on renvoie le programme à l'instruction GET dans une boucle continue jusqu'à ce qu'on appuie sur une touche du clavier. Quand on appuie sur une touche, le programme continue avec la ligne qui suit la ligne 10. Ajouter la ligne suivante au programme:

```
100 PRINT A$;: GOTO 10
```

Exécuter (RUN) maintenant le programme. On peut remarquer qu'aucun curseur ■ n'apparaît sur l'écran, mais les caractères tapés sont imprimés. On peut intégrer ce programme de deux lignes dans un programme éditeur d'écran, comme on le montre ci-dessous.

Un programme éditeur d'écran présente une grande flexibilité. Il permet d'avoir un curseur clignotant. On peut empêcher certaines touches, comme CLR/HOME, d'effacer accidentellement l'écran entier. On peut même vouloir utiliser les touches de fonction pour qu'elles représentent des expressions ou mots complets. Les lignes suivantes donnent aussi un but spécial à chaque touche de fonction. Il faut se rappeler que ce programme n'est qu'un début et qu'on peut l'adapter aux besoins de chacun.

```
20 IF A$ = CHR$(133) THEN POKE 53280,8:GOTO 10
30 IF A$ = CHR$(134) THEN POKE 53281,4:GOTO 10
40 IF A$ = CHR$(135) THEN A$ = "CHER MONSIEUR," + CHR$(13)
50 IF A$ = CHR$(136) THEN A$ = "CORDIALEMENT," + CHR$(13)
```

Les nombres CHR$ entre parenthèses sont tirés du tableau de codes CHR$ de l'annexe C. Le tableau donne un nombre différent pour chaque caractère. Les quatre touches de fonction sont réglées pour exécuter les tâches indiquées par les instructions qui suivent le mot THEN de chaque ligne. En changeant le nombre CHR$ dans chaque jeu de parenthèses, on peut désigner des touches différentes. On exécute des instructions différentes si on change les informations qui suivent l'instruction THEN.

"COMPRESSION" DES PROGRAMMES BASIC

On peut grouper davantage d'instructions (et de puissance) dans les programmes BASIC en les rendant les plus courts possible. Ce processus consiste à "comprimer" les programmes.

La compression des programmes permet d'intégrer un maximum d'instructions dans le programme. Elle permet aussi de réduire la taille de programmes qui pourraient être autrement difficiles à exécuter dans une taille donnée. Si l'on écrit un programme dans lequel on doit introduire des données comme des articles d'inventaire, des nombres ou du texte, un programme court permet de disposer de davantage de place de mémoire pour les données.

ABRÉVIATION DES MOTS CLÉS

L'annexe A donne une liste des abréviations des mots clés. Cette liste est très utile pendant la programmation, car les abréviations permettent de grouper davantage de renseignements dans chaque ligne. Le point d'interrogation (?) est l'abréviation la plus fréquemment utilisée; il correspond à l'abréviation BASIC de la commande PRINT. Si on liste (LIST) toutefois un programme contenant des abréviations, le Commodore 64 imprime automatiquement la liste avec les mots clés en longueur normale. Si une ligne de programme dépasse 80 caractères (2 lignes sur l'écran) avec les mots clés non abré-

gés et si on désire la changer, on doit la réintroduire avec les abréviations avant de sauvegarder le programme. La sauvegarde (SAVE) d'un programme inclut les mots clés sans rallonger les lignes, car les mots clés BASIC sont symbolisés par le Commodore 64. En général, les abréviations sont ajoutées quand un programme est écrit et qu'il ne doit plus être listé (LIST) avant d'être sauvegardé (SAVE).

COMPRESSION DES NUMÉROS DE LIGNE DE PROGRAMME

On commence la plupart des programmes à la ligne 100 et on numérote les lignes suivantes à intervalles de 10 (100, 110, 120, etc.). On peut ainsi insérer d'autres lignes d'instructions (111, 112, etc.) à mesure que le programme progresse. Quand le programme est terminé, *on peut le comprimer en adoptant des numéros de ligne plus bas* (par exemple, 1, 2, 3), car les numéros plus longs occupent davantage de mémoire que les numéros courts quand on utilise les instructions GOTO et GOSUB. Par exemple, le numéro 100 occupe 3 octets de mémoire (un pour chaque chiffre); le numéro 1 n'en utilise qu'un.

INSERTION D'INSTRUCTIONS MULTIPLES DANS CHAQUE LIGNE

On peut introduire plus d'une instruction dans chaque ligne numérotée d'un programme si on les sépare par un deux-points. Pour chaque ligne, les instructions, y compris le deux-points, ne doivent pas dépasser la longueur normale de 80 caractères. Nous donnons ci-dessous un exemple de deux programmes, avant et après la compression:

AVANT LA COMPRESSION:

```
10 PRINT "BONJOUR. . .";
20 FOR T=1 TO 500:NEXT
30 PRINT "BONJOUR, ENCORE. . ."
40 GOTO 10
```

APRÈS LA COMPRESSION:

```
10 PRINT "BONJOUR. . .";:FORT=1
TO 500:NEXT:PRINT"BONJOUR,
ENCORE. . .":GOTO10
```

SUPPRESSION DES INSTRUCTIONS REM

Les instructions REM sont pratiques pour rappeler à l'utilisateur et aux autres programmeurs le rôle d'une section donnée d'un programme. Toutefois, quand le programme est terminé et prêt à l'exécution, il est probable que l'on n'ait plus besoin des instructions REM; on peut économiser de la place de mémoire en les éliminant. Si l'on projette de réviser ou d'étudier la structure du programme par la suite, il est bon de conserver une copie du programme complet avec les instructions REM.

UTILISATION DES VARIABLES

Si l'on utilise à plusieurs reprises un nombre, un mot ou une phrase dans un programme, il est généralement préférable de définir les mots ou nombres longs par une variable d'une ou deux lettres. On peut définir des nombres par des lettres simples. On peut représenter les mots et les phrases par des variables en chaîne avec une lettre et le signe dollar. En voici un exemple:

AVANT LA COMPRESSION:

```
10 POKE 54296,15
20 POKE 54276,33
30 POKE 54273,10
40 POKE 54273,40
50 POKE 54273,70
60 POKE 54296,0
```

APRÈS LA COMPRESSION:

```
10 V = 54296:F = 54273
20 POKEV,15:POKE54276,33
30 POKEF,10:POKEF,40:POKEF,70
40 POKEV,0
```

UTILISATION DES INSTRUCTIONS READ ET DATA

On peut taper d'importantes quantités de données en un seul bloc à la fois à plusieurs reprises; on peut aussi imprimer une fois la partie des instructions du programme et imprimer la totalité des données à traiter dans une longue liste avec l'instruction DATA. Cette disposition est particulièrement utile pour grouper de longues listes de numéros dans un programme.

UTILISATION DES TABLEAUX ET DES MATRICES

Les tableaux et les matrices sont analogues aux instructions DATA; en effet, on peut traiter de grandes quantités de données sous la forme d'une liste avec la partie traitement des données du programme faisant usage de cette liste, en séquence. Dans les tableaux, la liste peut être multidimensionnelle.

ÉLIMINATION DES ESPACES

On peut facilement diminuer la taille d'un programme en éliminant tous les espaces. Aux fins de clarté, nous plaçons souvent des espaces dans les exemples de programme, mais ils ne sont en fait pas utiles; on économise de l'espace en les supprimant.

UTILISATION DES SOUS-PROGRAMMES GOSUB

Si l'on utilise à plusieurs reprises une instruction ou une ligne donnée, il peut être bon d'utiliser GOSUB pour y revenir à partir d'autres points du programme, plutôt que d'écrire la ligne ou instruction complète chaque fois que l'on en a besoin.

UTILISATION DES INSTRUCTIONS TAB ET SPC

Au lieu d'imprimer (PRINT) plusieurs commandes de curseur pour placer un caractère sur l'écran, il est souvent plus économique d'utiliser des instructions TAB et SPC pour disposer mots ou caractères sur l'écran.

CHAPITRE 2

VOCABULAIRE DU LANGAGE BASIC

- Introduction
- Mots clés, abréviations, et types de fonction en BASIC
- Description des mots clés en BASIC (ordre alphabétique)
- Clavier et caractéristiques du Commodore 64
- Éditeur d'écran

INTRODUCTION

Ce chapitre explique les ***mots clés du langage BASIC de CBM.*** Nous commençons avec une liste de mots clés faciles, accompagnés de leurs abréviations et de leur aspect sur l'écran. Nous expliquons ensuite la syntaxe et le rôle de chaque mot clé en détail. Des exemples donnent une idée de leur utilisation dans les programmes.

Le BASIC de Commodore 64 est pratique parce qu'il permet d'abréger la plupart des mots clés. Avec les abréviations, il faut introduire suffisamment de lettres d'un mot clé pour le distinguer des autres; on introduit le dernier caractère graphique ou lettre en appuyant sur la touche SHIFT .

Les abréviations n'économisent PAS de place de mémoire quand on les utilise dans les programmes, car tous les mots clés sont réduits à des symboles à un caractère par l'interpréteur BASIC. Quand on liste un programme contenant des abréviations, les mots clés apparaissent sous leur forme orthographique complète. On peut utiliser des abréviations pour placer davantage d'instructions dans une ligne de programme, même si elles n'entrent pas dans la ligne d'écran logique de 80 caractères. L'éditeur d'écran travaille avec une ligne de 80 caractères; si l'on utilise des abréviations dans une ligne qui dépasse 80 caractères, on ne peut PAS l'éditer quand on la liste (LIST). À la place, on doit (1) retaper la ligne entière, avec toutes les abréviations ou (2) diviser la ligne de codes en deux lignes, chacune ayant son propre numéro.

La table 2-1 donne une liste complète de mots clés avec leurs abréviations et leur aspect sur l'écran. Ce tableau est suivi d'une description alphabétique de toutes les instructions, commandes et fonctions permises avec le Commodore 64.

Ce chapitre explique également les fonctions BASIC intégrées dans l'interpréteur de langage BASIC. On peut utiliser les fonctions incorporées dans les instructions en mode direct ou dans tout programme, sans devoir les définir davantage. Il n'en est pas de même avec les fonctions définies par l'utilisateur. On peut utiliser les résultats des fonctions BASIC intégrées comme sortie immédiate ou les attribuer à un nom de variable de type approprié. Il existe deux types de fonction BASIC:

1) NUMÉRIQUE
2) CHAÎNE

Les arguments des fonctions intégrées sont toujours placés entre parenthèses ( ). Les parenthèses suivent toujours immédiatement le mot clé de fonction; il n'y a pas d'espace entre la dernière lettre du mot clé et la parenthèse de gauche (.

Le type de données du résultat détermine généralement le type d'argument nécessaire. Les fonctions qui retournent un résultat sous forme de valeur en chaîne sont identifiées par un mot clé dont le dernier caractère est un signe dollar ($). Dans certains cas, les fonctions en chaîne contiennent un ou plusieurs arguments numériques.

Les fonctions numériques peuvent se convertir du format entier au format à point flottant, suivant le cas. Dans les descriptions suivantes, le type de données de la valeur retournée est indiqué avec chaque nom de fonction. Les types d'argument sont également donnés avec le format de l'instruction.

Table 2-1. MOTS CLÉS DU BASIC DE COMMODORE 64

COMMANDE	ABRÉVIATION	ÉCRAN	TYPE DE FONCTION
ABS	A SHIFT B	A	NUMÉRIQUE
AND	A SHIFT N	A	
ASC	A SHIFT S	A ♥	NUMÉRIQUE
ATN	A SHIFT T	A	NUMÉRIQUE
CHR$	C SHIFT H	C	CHAÎNE
CLOSE	CL SHIFT O	CL	
CLR	C SHIFT L	C	
CMD	C SHIFT M	C	
CONT	C SHIFT O	C	
COS	aucune	COS	NUMÉRIQUE
DATA	D SHIFT A	D ♠	
DEF	D SHIFT E	D	
DIM	D SHIFT I	D	

COMMANDE	ABRÉVIATION	ÉCRAN	TYPE DE FONCTION
END	E [SHIFT] N	E ╱	
EXP	E [SHIFT] X	E ♣	NUMÉRIQUE
FN	aucune	FN	
FOR	F [SHIFT] O	F	
FRE	F [SHIFT] R	F	NUMÉRIQUE
GET	G [SHIFT] E	G	
GET#	aucune	GET#	
GOSUB	GO [SHIFT] S	GO ♥	
GOTO	G [SHIFT] O	G	
IF	aucune	IF	
INPUT	aucune	INPUT	
INPUT#	I [SHIFT] N	I ╱	
INT	aucune	INT	NUMÉRIQUE
LEFT$	LE [SHIFT] F	LE	CHAÎNE
LEN	aucune	LEN	NUMÉRIQUE
LET	L [SHIFT] E	L	
LIST	L [SHIFT] I	L	
LOAD	L [SHIFT] O	L	
LOG	aucune	LOG	NUMÉRIQUE

COMMANDE	ABRÉVIATION	ÉCRAN	TYPE DE FONCTION
MID$	M SHIFT I	M ◹	CHAÎNE
NEW	aucune	NEW	
NEXT	N SHIFT E	N ▔	
NOT	N SHIFT O	N ◻	
ON	aucune	ON	
OPEN	O SHIFT P	O ◻	
OR	aucune	OR	
PEEK	P SHIFT E	P ▔	NUMÉRIQUE
POKE	P SHIFT O	P ◻	
POS	aucune	POS	NUMÉRIQUE
PRINT	?	?	
PRINT#	P SHIFT R	P ▁	
READ	R SHIFT E	R ▔	
REM	aucune	REM	
RESTORE	RE SHIFT S	RE ♥	
RETURN	RE SHIFT T	RE ▏	
RIGHT$	R SHIFT I	R ◹	CHAÎNE
RND	R SHIFT N	R ╱	NUMÉRIQUE
RUN	R SHIFT U	R ◜	

COMMANDE	ABRÉVIATION	ÉCRAN	TYPE DE FONCTION
SAVE	S SHIFT A	S ♠	
SGN	S SHIFT G	S ┃	NUMÉRIQUE
SIN	S SHIFT I	S ╮	NUMÉRIQUE
SPC(	S SHIFT P	S ┐	SPÉCIAL
SQR	S SHIFT Q	S ●	NUMÉRIQUE
STATUS	ST	ST	NUMÉRIQUE
STEP	ST SHIFT E	ST ─	
STOP	S SHIFT T	S ┃	
STR$	ST SHIFT R	ST ▁	CHAÎNE
SYS	S SHIFT Y	S ┃	
TAB(	T SHIFT A	T ♠	SPÉCIAL
TAN	aucune	TAN	NUMÉRIQUE
THEN	T SHIFT H	T ┃	
TIME	TI	TI	NUMÉRIQUE
TIME$	TI$	TI$	CHAÎNE
TO	aucune	TO	
USR	U SHIFT S	U ♥	NUMÉRIQUE
VAL	V SHIFT A	V ♠	NUMÉRIQUE
VERIFY	V SHIFT E	V ▔	
WAIT	W SHIFT A	W ♠	

DESCRIPTION DES MOTS CLÉS DE BASIC

ABS

TYPE: Fonction numérique
FORMAT: ABS(<expression>)

Rôle: Retourne la valeur absolue du nombre, c'est-à-dire sa valeur sans signe. La valeur absolue d'un nombre négatif s'obtient en le multipliant par −1.

EXEMPLES de fonction ABS:

```
10 X = ABS ( Y )
10 PRINT ABS ( X * J )
10 IF X = ABS (X) THEN PRINT "POSITIVE"
```

AND

TYPE: Opérateur
FORMAT: <expression> AND <expression>

Rôle: AND s'utilise dans les opérations booléenes pour vérifier les bist. Dans les opérations, il sert aussi à vérifier l'exactitude des facteurs.

En algèbre de Boole, le résultat d'une opération AND n'est 1 que si ses deux facteurs sont 1. Le résultat est 0 si l'un des facteurs ou les deux sont 0 (faux).

EXEMPLES d'opération AND à 1 bit:

0	1	0	1
AND 0	AND 0	AND 1	AND 1
0	0	0	1

Le Commodore 64 exécute l'opération AND sur les nombres de l'intervalle −32768 à +32767. Il n'est pas utilisé de valeurs fractionnaires et les nombres en dehors de l'intervalle amènent une erreur **?ILLEGAL QUANTITY** (quantité interdite). Converti en format binaire, cet intervalle donne 16 bits pour chaque nombre. Les bits correspondants, après l'opération AND, donnent un résultat de 16 bits dans le même intervalle.

EXEMPLES d'opérations AND à 16 bits:

```
                                   17
                              AND 194
                      ---------------
                     0000000000010001
                 AND 0000000011000010
-------------------------------------
   (BINAIRE) 0000000000000000
-------------------------------------
(DÉCIMAL)                           0

                                32007
                            AND 28761
                      ---------------
                     0111110100000111
                 AND 0111000001011001
-------------------------------------
   (BINAIRE) 0111000000000001
-------------------------------------
(DÉCIMAL)                       28673

                                 -241
                            AND 15359
                      ---------------
                     1111111100001111
                 AND 0011101111111111
-------------------------------------
   (BINAIRE) 0011101100001111
-------------------------------------
(DÉCIMAL)                       15119
```

Quand on évalue l'exactitude ou l'inexactitude d'un nombre, l'ordinateur suppose qu'il est exact tant que sa valeur diffère de 0. Quand il évalue une comparaison, il attribue une valeur de −1 si le résultat est vrai et de 0 s'il est faux. En format binaire, −1 donne des 1 et 0 des 0. De ce fait, si l'on combine les évaluations vraies/fausses dans les opérations AND, le résultat est vrai s'il contient des bits vrais.

EXEMPLES d'utilisation de l'opération AND avec des évaluations vraies/fausses:

```
50 IF X=7 AND W=3 THEN GOTO 10:REM VRAI SEULEMENT SI X=7
   ET W=3 SONT VRAIS
60 IF A AND Q=7 THEN GOTO 10:REM VRAI SI A N'EST PAS NUL
   ET Q=7 EST VRAI
```

ASC

TYPE: Fonction numérique
FORMAT: ASC (<chaîne>)

Rôle: ASC retourne un nombre de 0 à 255 qui correspond à la valeur ASCII Commodore du premier caractère de la chaîne. L'annexe C donne la table des valeurs ASCII Commodore.

EXEMPLES de fonction ASC:

```
10 PRINT ASC("Z")
20 X = ASC("ZEBRA")
30 J = ASC(J$)
```

S'il n'y a aucun caractère dans la chaîne, on obtient un message d'erreur **?ILLEGAL QUANTITY** (quantité interdite). Dans le troisième exemple ci-dessus, si J$="", la fonction ASC ne marche pas. Pour l'instruction GET ou GET#, CHR$(0) est une chaîne vide. Pour éliminer cet inconvénient, ajouter CHR$(0) à la fin d'une chaîne de la façon indiquée ci-dessous.

EXEMPLE de fonction ASC évitant une ERREUR DE QUANTITÉ INTERDITE:

```
30 J = ASC(J$ + CHR$(0))
```

ATN

TYPE: Fonction numérique
FORMAT: ATN (< nombre >)

Rôle: Cette fonction mathématique retourne l'arc tangente du nombre. Le résultat est l'angle en radians dont la tangente est le nombre indiqué. Le résultat est toujours compris dans l'intervalle $-\pi/2$ à $+\pi/2$.

EXEMPLES de fonction ATN:

```
10 PRINT ATN ( 0 )
20 X = ATN ( J ) * 180 / π : REM CONVERSION EN DEGRÉS
```

CHR$

TYPE: Fonction-chaîne
FORMAT: CHR$ (< nombre >)

Rôle: Cette fonction convertit un code ASCII Commodore dans son équivalent en caractères. L'annexe C donne une liste des caractères et de leurs codes. Le nombre doit être compris entre 0 et 255 sinon, on obtient un message d'erreur **?ILLEGAL QUANTITY** (quantité interdite).

EXEMPLES de fonction CHR$:

```
10 PRINT CHR$(65) : REM 65 = MAJUSCULE A
20 A$ = CHR$(13) : REM 13 = TOUCHE RETURN
50 A = ASC(A$) : A$ = CHR$(A): REM CONVERTIT EN CODE ASCII
     DE C64 ET VICE VERSA
```

CLOSE

TYPE: Instruction d'entrée/sortie
FORMAT: CLOSE <numéro de fichier>

Rôle: Cette instruction interrompt tout fichier ou canal de données vers un dispositif. Le numéro de fichier est identique à celui utilisé à l'ouverture (OPEN) du fichier ou du dispositif (voir instruction OPEN et la section sur la programmation d'entrée/sortie).

Si l'on travaille avec des dispositifs de stockage comme les cassettes et les disques, l'opération CLOSE stocke les tampons incomplets dans le dispositif. Si l'on n'exécute pas cette opération, le fichier est incomplet sur cassette et illisible sur le disque. L'opération CLOSE n'est pas aussi cruciale avec les autres dispositifs, mais elle libère de la mémoire pour d'autres fichiers. Pour plus de détails, consulter le manuel du dispositif extérieur.

EXEMPLES d'instruction CLOSE:

```
10 CLOSE 1
20 CLOSE X
30 CLOSE 9 * ( 1 + J )
```

CLR

TYPE: Instruction
FORMAT: CLR

Rôle: Cette instruction libère de la mémoire vive (RAM) qui a été utilisée mais n'est plus nécessaire. Les programmes BASIC en mémoire restent intacts, mais les variables, tableaux, adresses GOSUB, boucles FOR. . .NEXT, fonctions définies par l'utilisateur et fichiers sont effacés de la mémoire; leur espace peut maintenant s'utiliser avec de nouvelles variables, etc.

Les fichiers sur disque et sur cassette ne sont pas convenablement fermés (CLOSE) par l'instruction CLR. Les renseignements sur les fichiers sont perdus dans l'ordinateur, y compris les tampons incomplets. Pour l'unité de disques, le fichier est toujours ouvert (OPEN). Pour plus de détails à ce sujet, voir l'instruction CLOSE.

EXEMPLE d'instruction CLR:

```
10 X=25
20 CLR
30 PRINT X

RUN
0

READY
```

CMD

TYPE: Instruction d'entrée/sortie
FORMAT: CMD <numéro de fichier> [,chaîne]

Rôle: Cette instruction fait passer le dispositif de sortie primaire de l'écran du téléviseur au fichier spécifié. Le fichier peut être sur disque ou cassette; il peut aussi correspondre à une imprimante ou à un dispositif d'entrée/sortie comme le modem. On doit préciser le numéro de fichier dans une instruction OPEN précédente. La chaîne précisée est envoyée au fichier. Cette opération est pratique pour donner des titres aux sorties d'imprimante, etc.

Quand cette commande est en service, les instructions PRINT et les commandes LIST ne s'affichent pas sur l'écran, mais elles envoient le texte au fichier, sous le même format.

Pour ramener la sortie à l'écran, envoyer une ligne vide avec la commande PRINT# au dispositif CMD avant de clore (CLOSE) pour arrêter les données en attente ("désélection" du dispositif).

Toute erreur de système, comme **?SYNTAX ERROR** (erreur de syntaxe), amène le retour de la sortie à l'écran. Les dispositifs ne sont pas "désélectionnés"; il faut donc envoyer une ligne vide après une condition d'erreur. (Pour plus de détails, consulter le manuel de l'imprimante ou de l'unité de disques.)

EXEMPLES d'instruction CMD:

```
OPEN 4, 4: CMD 4, "TITRE": LIST: REM LISTE LE PROGRAMME SUR
    L'IMPRIMANTE
PRINT# 4: CLOSE 4: REM DÉSÉLECTIONNE ET COUPE L'IMPRIMANTE

10 OPEN 1, 1, 1, "ESSAI": REM CRÉATION D'UN FICHIER SÉQUENTIEL
20 CMD 1: REM SORTIE VERS FICHIER CASSETTE, NON VERS ÉCRAN
30 FOR L = 1 TO 100
40 PRINT L: REM MET LE NOMBRE DANS TAMPON DE CASSETTE
50 NEXT
60 PRINT# 1: REM DÉSÉLECTION
70 CLOSE 1: REM ÉCRITURE DU TAMPON NON FINI, FIN CORRECTE
```

CONT

TYPE: Commande
FORMAT: CONT

Rôle: Cette commande reprend l'exécution d'un programme qui avait été interrompu par une instruction STOP ou END ou par une pression sur la touche RUN/STOP. Le programme reprend au point exact d'interruption.

Quand le programme est arrêté, l'utilisateur peut examiner ou changer les variables ou inspecter le programme. Pendant la mise au point ou l'examen d'un programme, on peut introduire des instructions STOP aux points stratégiques pour permettre l'examen des variables et vérifier le déroulement du programme.

On obtient le message d'erreur **CAN'T CONTINUE** (impossible de continuer) si l'on édite le programme (même si l'on appuie sur RETURN avec le curseur sur une ligne non changée), si le programme s'est interrompu du fait d'une erreur ou si l'on a introduit une erreur avant d'avoir tapé CONT pour reprendre le programme.

EXEMPLE de commande CONT:

```
10 PI=0:C=1
20 PI=PI+4/C-4/(C+2)
30 PRINT PI
40 C=C+4:GOTO 20
```

Ce programme calcule la valeur de PI. L'exécuter (RUN) et, après quelques instants, appuyer sur la touche RUN/STOP . On obtient l'affichage:

BREAK IN 20

REMARQUE: Le nombre peut être différent.

Taper la commande PRINT C pour observer les progrès du Commodore 64. Taper ensuite CONT pour reprendre au point d'interruption.

COS

TYPE: Fonction
FORMAT: COS (<nombre>)

Rôle: Cette fonction mathématique calcule le cosinus du nombre, celui-ci étant un angle en radians.

EXEMPLES de fonction COS:

```
10 PRINT COS ( 0 )
20 X = COS (Y * π/ 180): REM CONVERSION DES DEGRÉS EN RADIANS
```

DATA

TYPE: Instruction
FORMAT: DATA <liste des constantes>

Rôle: Les instructions DATA stockent les informations dans un programme. Celui-ci utilise les informations à l'aide de l'instruction READ qui extrait les constantes successives des instructions DATA.

Les instructions DATA n'ont pas à être exécutées par le programme; il suffit qu'elles soient présentes. De ce fait, elles sont généralement placées à la fin du programme.

Dans un programme, toutes les instructions de données sont traitées en liste continue. Les données sont lues (READ) de gauche à droite, de la ligne de numéro le plus bas à la ligne de numéro le plus élevé. Si l'instruction READ rencontre des données ne correspondant pas au type demandé (par exemple, elle cherche un nombre et trouve une chaîne), il se produit un message d'erreur.

Des caractères peuvent servir de données, mais si certains sont utilisés, l'élément de donnée doit être placé entre guillemets (" "). Ces caractères comprennent les signes de ponctuation comme la virgule (,), les deux points (:), les espaces vierges, les lettres majuscules/minuscules, les caractères graphiques et les caractères de commande de curseur.

EXEMPLES d'instruction DATA:

```
10 DATA 1, 10, 5, 8
20 DATA JOHN, PAUL, GEORGE, RINGO
30 DATA "CHÈRE HÉLÈNE, COMMENT VAS-TU, AMITIÉS, PIERRE"
40 DATA -1.7E-9, 3.33
```

DEF FN

TYPE: Instruction
FORMAT: DEF FN <nom> (<variable>) = <expression>

Rôle: Cette instruction établit une fonction définie par l'utilisateur dont on se servira par la suite dans le programme. La fonction peut se composer d'une formule mathématique. Quand une longue formule est utilisée à plusieurs endroits, les fonctions définies par l'utilisateur permettent de conserver de la place dans les programmes. Il suffit de préciser la formule une seule fois, dans l'instruction de définition, et de l'abréger ensuite en nom de fonction. On doit l'exécuter une fois, mais les exécutions ultérieures sont omises.

Le nom de la fonction correspond aux lettres FN suivies d'un nom de variable qui peut se composer de 1 ou 2 caractères, le premier étant une lettre et le deuxième un chiffre ou une lettre.

EXEMPLES d'instruction DEF FN:

```
10 DEF FN A(X) = X + 7
20 DEF FN AA(X) = Y * Z
30 DEF FNA9 (Q) = INT( RND( 1)* Q+ 1)
```

Pour appeler la fonction dans la suite du programme, on utilise le nom de la fonction avec une variable entre parenthèses. Ce nom de fonction s'utilise comme toute autre variable et sa valeur se calcule automatiquement.

EXEMPLES d'utilisation de FN:

40 PRINT FN A (9)
50 R = FNAA (9)
60 G = G + FN A9 (10)

À la ligne 50 ci-dessus, le chiffre 9 entre parenthèses n'a aucun effet sur le résultat de la fonction, car la définition de la ligne 20 n'utilise pas la variable entre parenthèses. Quelle que soit la valeur de X, le résultat est Y fois Z. Dans les deux autres fonctions, la valeur entre parenthèses influe sur le résultat.

DIM

TYPE: Instruction
FORMAT: DIM <variable> (<indices inférieurs>) [, <variable> (<indices inférieurs>) . . .]

Rôle: Cette instruction définit un tableau ou une matrice de variables. Elle permet d'utiliser le nom de variable avec un indice inférieur. L'indice inférieur précise l'élément utilisé. Dans un tableau, le numéro d'élément le plus bas est zéro; le plus élevé est le numéro indiqué dans l'instruction DIM (maximum de 32767).

On doit exécuter l'instruction DIM une fois, *et une fois uniquement*, pour chaque tableau. Si l'on exécute de nouveau cette ligne, on obtient une erreur **REDIM'D ARRAY** (tableau redimensionné). De ce fait, la plupart des programmes exécutent toutes les opérations DIM dès le début.

Dans un tableau, on peut avoir n'importe quel nombre de dimensions et 255 indices inférieurs, la seule limite étant l'espace de mémoire vive (RAM) disponible pour recevoir les variables. Le tableau peut se composer de variables numériques normales, comme on l'indique ci-dessus, ou de chaînes ou de nombres entiers. En dehors des variables numériques normales, utiliser le signe $ ou % après le nom pour indiquer les variables entières ou en chaîne.

Si un tableau cité dans un programme n'a jamais été dimensionné (DIM), il l'est automatiquement à 11 éléments dans chaque dimension utilisée dans la première indication.

EXEMPLES d'instruction DIM:

```
10 DIM A ( 100 )
20 DIM Z ( 5, 7), Y ( 3, 4, 5)
30 DIM Y7% ( Q )
40 DIM PH$ (1000)
50 F (4) =9: REM EXÉCUTION AUTOMATIQUE DE DIM F (10)
```

EXEMPLE DE MARQUE DE FOOTBALL AVEC DIM:

```
10 DIM S(1,5), T$(1)
20 INPUT "NOMS DES ÉQUIPES";T$(0),T$(1)
30 FOR Q=1 TO 5: FOR T=0 TO 1
40 PRINT T$(T), "MARQUE DANS LA PÉRIODE" Q
50 INPUT S(T,Q): S(T,0)= S(T,O)+ S(T,Q)
60 NEXT T,Q
70 PRINT CHR$(147) "TABLEAU DE MARQUE"
80 PRINT "PÉRIODE"
90 FOR Q=1 TO 5
100 PRINT TAB( Q*2 +9) Q;
110 NEXT: PRINT TAB(15) "TOTAL"
120 FOR T=0 TO 1: PRINT T$(T);
130 FOR Q=1 TO 5
140 PRINT TAB(Q*2 +9) S(T,Q);
150 NEXT: PRINT TAB(15) S(T,O)
160 NEXT
```

CALCUL DE LA MÉMOIRE UTILISÉE PAR L'INSTRUCTION DIM:

5 octets pour le nom du tableau
2 octets pour chaque dimension
2 octets par élément pour les variables entières
5 octets par élément pour les variables numériques normales
3 octets par élément pour les variables en chaîne
1 octet pour chaque caractère de chaque élément de chaîne

END

TYPE: Instruction
FORMAT: END

Rôle: Cette instruction termine l'exécution d'un programme et affiche le message READY (prêt) puis retourne la commande à l'utilisateur de l'ordinateur. Un programme peut contenir plusieurs instructions END. Il n'est pas essentiel d'inclure des instructions END dans un programme, mais on recommande cependant de le terminer par une instruction END.

L'instruction END est analogue à l'instruction STOP. STOP amène cependant l'ordinateur à afficher le message **BREAK IN LINE XX** (interruption à la ligne XX) et END affiche READY (prêt). Avec les deux instructions, l'ordinateur peut reprendre l'exécution si on tape la commande CONT.

EXEMPLES d'instruction END:

```
10 PRINT "VOULEZ-VOUS VRAIMENT EXÉCUTER CE PROGRAMME"
20 INPUT A$
30 IF A$ = "NON" THEN END
40 REM RESTE DU PROGRAMME . . .
999 END
```

EXP

TYPE: Fonction numérique
FORMAT: EXP (<nombre>)

Rôle: Cette fonction mathématique calcule la constante e (2.71828183) élevée à la puissance indiquée. Une valeur supérieure à 88.0296919 provoque une erreur **?OVERFLOW** (dépassement de capacité).

EXEMPLES de fonction EXP:

```
10 PRINT EXP (1)
20 X = Y * EXP (Z * Q)
```

FN

TYPE: Fonction numérique
FORMAT: FN <nom> (<nombre>)

Rôle: Cette fonction désigne la formule précédemment définie (DEF) par son nom. Le nombre remplace le nom (le cas échéant) et la formule est calculée. Le résultat est une valeur numérique.

On peut utiliser cette fonction en mode direct pourvu que l'instruction DEF ait été exécutée.

Si une fonction FN est exécutée avant l'instruction DEF qui la définit, on obtient une erreur **UNDEF'D FUNCTION** (fonction non définie).

EXEMPLES de fonction FN (définie par l'utilisateur)

```
PRINT FN A ( Q )
1100 J = FN J (7) + FN J (9)
9990 IF FN B7 (I+1)= 6 THEN END
```

FOR . . . TO . . . [STEP . . .]

TYPE: Instruction
FORMAT: FOR <variable> = <début> TO <limite> [STEP <valeur de progression>]

Rôle: Cette instruction BASIC spéciale permet d'utiliser une variable comme compteur. On doit préciser certains paramètres comme le nom de la variable à point flottant, sa valeur de départ, la limite du comptage et la valeur à ajouter pendant chaque cycle.

Nous donnons ci-dessous un exemple de programme BASIC simple qui compte de 1 à 10, imprime (PRINT) chaque chiffre et se termine (END), sans utilisation de l'instruction FOR:

```
100 L = 1
110 PRINT L
120 L = L + 1
130 IF L < = 10 THEN 110
140 END
```

Nous donnons ci-dessous le même programme, avec l'instruction FOR:

```
100 FOR L = 1 TO 10
110 PRINT L
120 NEXT L
130 END
```

On voit que le programme est plus court et plus facile à comprendre avec l'instruction FOR.

Quand l'instruction FOR est exécutée, il se produit plusieurs opérations. La valeur de <début> est placée dans la <variable> utilisée dans le compteur. Dans l'exemple ci-dessus, un 1 est placé dans L.

Quand on arrive à l'instruction NEXT, la <valeur de progression> est ajoutée à la <variable>. Si l'on n'a pas inclus STEP, la <valeur de progression> est fixée à +1. Quand le programme ci-dessus arrive à la ligne 120 pour la première fois, 1 est ajouté à L dont la nouvelle valeur devient 2.

La valeur dans la <variable> est maintenant comparée à la <limite>. Si le programme n'a pas encore atteint cette <limite>, le programme va (GOTO) à la ligne qui suit l'instruction FOR initiale. Dans ce cas, la valeur de 2 dans L est inférieure à la limite 10; le programme va donc (GOTO) à la ligne 110.

La <variable> finit par dépasser la valeur de <limite>. À ce moment, la boucle se termine et le programme passe à la ligne qui suit l'instruction NEXT. Dans notre exemple, la valeur L atteint 11 qui dépasse donc la limite 10; le programme passe alors à la ligne 130.

Si la <valeur de progression> est positive, la <variable> doit dépasser la <limite>; si elle est négative, elle doit devenir inférieure à la <limite>.

REMARQUE: Une boucle s'exécute toujours au moins une fois.

EXEMPLES d'instruction FOR . . . TO . . . STEP . . . :

```
100 FOR L = 100 TO 0 STEP -1
100 FOR L = PI TO 6* π STEP .01
100 FOR AA = 3 TO 3
```

FRE

TYPE: Fonction
FORMAT: FRE (<variable>)

Rôle: Cette fonction indique l'espace de mémoire vive (RAM) disponible pour le programme et ses variables. Si un programme essaie d'utiliser plus d'espace qu'il n'en reste, on obtient le message **OUT OF MEMORY** (mémoire pleine).

La valeur du nombre entre parenthèses n'a pas d'importance, car elle ne sert pas dans le calcul.

> **REMARQUE:** Si le résultat de la fonction FRE est négatif, ajouter 65536 au nombre FRE pour obtenir le nombre d'octets de mémoire disponibles.

EXEMPLES de fonction FRE:

```
PRINT FRE ( 0 )
10 X = ( FRE ( K ) - 1000 )/7
950 IF FRE ( 0 ) < 100 THEN PRINT "PAS ASSEZ DE PLACE"
```

> **REMARQUE:** La ligne suivante indique toujours la place de mémoire vive (RAM) disponible:
>
> ```
> PRINT FRE(0) - (FRE(0) < 0)* 65536
> ```

GET

TYPE: Instruction
FORMAT: GET <liste de variables>

Rôle: Cette instruction lit chaque frappe de touche de l'utilisateur. Quand l'utilisateur tape, les caractères sont stockés dans la mémoire tampon du Commodore 64. Cette mémoire peut recevoir jusqu'à 10 caractères; les touches frappées après la 10e sont perdues. La lecture d'un des caractères avec l'instruction GET fait de la place pour un autre caractère.

Si l'instruction GET précise des données numériques et si l'utilisateur ne tape pas sur une touche de chiffre, on obtient le message **?SYNTAX ERROR** (erreur de syntaxe). Pour plus de sûreté, lire les touches sous forme de chaînes et les convertir en nombres par la suite.

On peut utiliser l'instruction GET pour éviter certaines limites de l'instruction INPUT. Pour plus de détails à ce sujet, voir la section "Utilisation de l'instruction GET" dans les techniques de programmation.

EXEMPLES d'instruction GET:

```
10 GET A$:IF A$ = "" THEN 10: REM BOUCLE SUR 10 JUSQU'A CE QU'ON
     FRAPPE UNE TOUCHE
20 GET A$, B$, C$, D$, E$: REM LIT 5 TOUCHES
30 GET A, A$
```

GET#

TYPE: Instruction d'entrée/sortie
FORMAT: GET# <numéro de fichier>, <liste de variables>

Rôle: Cette instruction lit un par un les caractères du dispositif ou fichier spécifié. Elle joue le même rôle que l'instruction GET, mais les données ne viennent pas du clavier. S'il n'est pas reçu de caractères, la variable est fixée à une chaîne vide (égale à "") ou à 0 pour les variables numériques. Les caractères utilisés pour la séparation des données dans les fichiers, tels que la virgule (,) ou le code de touche RETURN (code ASC 13), sont reçus comme tout autre caractère.

Utilisée avec le dispositif 3 (écran de visualisation), cette instruction lit les caractères de l'écran un par un. Chaque utilisation de GET# déplace le curseur d'une position vers la droite. Le caractère qui termine la ligne logique est changé en CHR$(13), code de la touche RETURN.

EXEMPLES d'instruction GET#:

```
5 GET# 1, A$
10 OPEN 1, 3: GET# 1, Z7$
20 GET# 1, A, B, C$, D$
```

GOSUB

TYPE: Instruction
FORMAT: GOSUB <numéro de ligne>

Rôle: Cette instruction, forme spécialisée de GOTO, présente cependant une différence importante: GOSUB se rappelle sa position d'origine. Quand le programme atteint l'instruction RETURN (différente de la touche RETURN du clavier), il saute à l'instruction qui suit immédiatement l'instruction GOSUB originale.

On utilise surtout un sous-programme (par GOSUB, on entend passer (GO) à un sous-(SUB)programme) quand on emploie une petite section de programme dans différentes parties d'un programme plus important. En utilisant des sous-programmes au lieu de répéter des lignes identiques à plusieurs reprises en différents points d'un programme, on peut économiser beaucoup de place de mémoire. GOSUB est ainsi analogue à DEF FN qui permet d'économiser de la place dans l'utilisation d'une formule. GOSUB permet de conserver de l'espace dans le cas d'un sous-programme de plusieurs lignes. Dans le programme peu pratique ci-dessous, on n'utilise pas GOSUB:

```
100 PRINT "CE PROGRAMME IMPRIME"
110 FOR L = 1 TO 500 : NEXT
120 PRINT "LENTEMENT SUR L'ÉCRAN"
130 FOR L = 1 TO 500 : NEXT
140 PRINT "AVEC UNE BOUCLE SIMPLE"
150 FOR L = 1 TO 500 : NEXT
160 PRINT "POUR LA TEMPORISATION."
170 FOR L = 1 TO 500 : NEXT
```

Voici le même programme, avec l'instruction GOSUB:

```
100 PRINT "CE PROGRAMME IMPRIME"
110 GOSUB 200
120 PRINT "LENTEMENT SUR L'ÉCRAN"
130 GOSUB 200
140 PRINT "AVEC UNE BOUCLE SIMPLE"
150 GOSUB 200
160 PRINT "POUR LA TEMPORISATION"
170 GOSUB 200
180 END
200 FOR L = 1 TO 500 : NEXT
210 RETURN
```

Chaque fois que le programme exécute une instruction GOSUB, le numéro de ligne et la position dans la ligne de programme sont sauvegardés dans un endroit spécial dit "pile", qui occupe 256 octets de la mémoire. Cette capacité limite la quantité de données pouvant être stockées dans la pile. De ce fait, le nombre d'adresses de retour de sous-programme pouvant être stockées est limité; il faut veiller à ce que chaque instruction GOSUB tombe sur l'instruction RETURN correspondante sinon, on tombe à cours de mémoire, même s'il reste beaucoup d'octets libres.

GOTO

TYPE: Instruction
FORMAT: GOTO <numéro de ligne>
ou GO TO <numéro de ligne>

Rôle: Cette instruction permet au programme BASIC de ne pas exécuter les lignes dans l'ordre numérique. Le mot GOTO suivi d'un numéro fait sauter à la ligne correspondant à ce numéro. GOTO non suivi d'un numéro équivaut à GOTO 0. Le numéro de ligne doit suivre le mot GOTO.

GOTO permet de créer des boucles sans fin. À titre d'exemple très simple, citons une ligne qui "tourne" (GOTO) sur elle-même, comme 10 GOTO 10. On peut arrêter ce genre de boucle avec la touche **RUN/STOP** du clavier.

EXEMPLES d'instruction GOTO:

```
GOTO 100
10 GO TO 50
20 GOTO 999$
```

IF . . . THEN . . .

TYPE: Instruction
FORMAT: IF <expression> THEN <numéro de ligne>
IF <expression> GOTO <numéro de ligne>
IF <expression> THEN <instructions>

Rôle: C'est à cette instruction que le BASIC doit une grande partie de son "intelligence" ou possibilité d'évaluer des conditions et de prendre différentes mesures dépendant de l'issue.

Le mot IF est suivi d'une expression qui peut comprendre des variables, des chaînes, des nombres, des comparaisons et des opérateurs logiques. Le mot THEN, qui apparaît sur *la même ligne*, est suivi d'un numéro de ligne ou d'une ou plusieurs instructions BASIC. Si l'expression est fausse, tous les éléments qui suivent THEN sur la ligne sont ignorés et l'exécution se poursuit à la ligne suivante du programme. Avec un résultat vrai, le programme se branche sur le numéro de ligne qui suit le mot THEN ou exécute les autres instructions BASIC présentes sur cette ligne.

EXEMPLE d'instruction IF . . . GOTO . . . :

```
100 INPUT "TAPER UN NOMBRE"; N
110 IF N < = 0 GOTO 200
120 PRINT "RACINE CARRÉE=" SQR(N)
130 GOTO 100
200 PRINT "LE NOMBRE DOIT ÊTRE >0"
210 GOTO 100
```

Ce programme imprime la racine carrée de tout nombre positif. L'instruction IF sert ici à valider le résultat de l'entrée (INPUT). Si le résultat de N < = 0 est vrai, le programme saute à la ligne 200; si le résultat est faux, 120 est la ligne suivante exécutée. On peut remarquer que THEN GOTO n'est pas utile avec IF . . . THEN, car à la ligne 110, GOTO 200 signifie en réalité THEN GOTO 200.

EXEMPLE d'instruction IF . . . THEN . . . :

```
100 FOR L = 1 TO 100
110 IF RND(1) < .5 THEN X = X+ 1 : GOTO 130
120 Y = Y+ 1
130 NEXT L
140 PRINT "PILE= " X
150 PRINT "FACE= " Y
```

À la ligne 110, IF vérifie si un nombre au hasard est inférieur à .5. Si le résultat est vrai, la série complète d'instructions qui suivent le mot THEN est exécutée: X est d'abord augmenté de 1 puis le programme passe à la ligne 130. Si le résultat est faux, le programme saute à l'instruction suivante, à la ligne 120.

INPUT

TYPE: Instruction
FORMAT: INPUT [" <message-guide> ";] <liste de variables>

Rôle: Cette instruction permet à l'utilisateur qui exécute (RUN) le programme "d'injecter" des informations dans l'ordinateur. Quand on exécute cette instruction, elle imprime un point d'interrogation (?) sur l'écran et place le curseur à un espace à droite du point d'interrogation. Avec le curseur clignotant, l'ordinateur attend que l'utilisateur tape la réponse et appuie sur la touche RETURN .

Le mot INPUT peut être suivi d'un texte placé entre guillemets (" "). Ce texte est imprimé sur l'écran, suivi du point d'interrogation.

Le texte est suivi d'un point-virgule (;) et du nom d'une ou plusieurs variables, séparées par des virgules. L'ordinateur stocke dans cette variable les informations tapées par l'utilisateur. La variable peut se constituer d'un nom de variable permis. On peut avoir plusieurs noms de variable différents, chacun correspondant à une entrée différente.

EXEMPLES d'instruction INPUT:

```
100 INPUT A
110 INPUT B, C, D
120 INPUT "message-guide";E
```

Pendant l'exécution de ce programme, le point d'interrogation apparaît pour demander à l'utilisateur de donner une entrée à la ligne 100. Tout nombre tapé va dans A pour être utilisé ultérieurement dans le programme. Si la réponse tapée n'est pas un nombre, le message **?REDO FROM START** (reprendre du début) apparaît; le programme a en effet reçu une chaîne à la place d'un nombre. Si l'utilisateur a appuyé sur RETURN sans rien taper, la valeur de la variable ne change pas.

Un autre point d'interrogation apparaît pour la ligne 110. Si l'on ne tape qu'un nombre et si l'on appuie sur la touche RETURN, le Commodore 64 affiche alors 2 points d'interrogation; l'ordinateur veut une entrée plus complète. On peut taper autant d'entrées, séparées par des virgules, qu'on le désire pour éviter de faire apparaître le double point d'interrogation. Si l'on tape plus de données que l'instruction INPUT n'en demande, le message **?EXTRA IGNORED** (données en plus omises) apparaît; les éléments que l'on a tapés en trop n'ont pas été placés dans des variables.

La ligne 120 affiche "message-guide" avant le point d'interrogation. Le point-virgule est nécessaire entre le message-guide et toute liste de variables.

On ne peut jamais utiliser l'instruction INPUT en dehors d'un programme. Le Commodore 64 a besoin de mémoire tampon pour les variables INPUT (espace identique à celui utilisé pour les commandes).

INPUT#

TYPE: Instruction d'entrée/sortie
FORMAT: INPUT# <numéro de fichier> , <liste de variables>

Rôle: Cette instruction est particulièrement pratique et rapide pour extraire des données stockées dans un fichier sur disque ou sur cassette. Les données se présentent sous forme de variables complètes atteignant 80 caractères de longueur, au contraire de la méthode caractère par caractère de GET#. Le fichier doit d'abord avoir été ouvert (OPEN). INPUT# peut ensuite remplir les variables.

INPUT# suppose une variable terminée quand elle lit un code de retour (CHR$ (13)), une virgule (,), un point-virgule (;) ou deux points (:). On peut placer ces caractères entre guillemets s'ils sont nécessaires pendant l'écriture du programme (voir instructions PRINT#).

Si le type de variable utilisé est numérique et si l'on reçoit des caractères non numériques, on obtient une erreur **BAD DATA** (données erronées). INPUT# peut lire des chaînes atteignant 80 caractères; au-delà de cette longueur, on obtient une erreur **STRING TOO LONG** (chaîne trop longue).

Utilisée avec le dispositif 3 (écran de visualisation), cette instruction lit une ligne logique entière et déplace le curseur à la ligne suivante.

EXEMPLES d'instruction INPUT#:

```
10 INPUT# 1, A
20 INPUT# 2, A$, B$
```

INT

TYPE: Fonction entière
FORMAT: INT (<numérique>)

Rôle: Donne la valeur entière de l'expression. Si l'expression est positive, la partie fractionnaire est omise. Si l'expression est négative, toute partie fractionnaire fait apparaître la valeur entière inférieure suivante.

EXEMPLES de fonction INT:

```
120 PRINT INT(99.4343), INT(−12.34)

99          -13
```

LEFT$

TYPE: Fonction de chaîne
FORMAT: LEFT$ (<chaîne>, <nombre entier>)

Rôle: Donne une chaîne se composant du <nombre entier> de caractères les plus à gauche de la <chaîne>. La valeur du nombre entier doit se situer dans l'intervalle 0 à 255. Si le nombre entier est supérieur à la longueur de la chaîne, celle-ci est donnée en entier. Si un <nombre entière> zéro est utilisé, on obtient alors une chaîne vide (de longueur zéro).

EXEMPLES de fonction LEFT$:

```
10 A$ = "ORDINATEURS COMMODORE"
20 B$ = LEFT$(A$,11): PRINT B$
RUN

ORDINATEURS
```

LEN

TYPE: Fonction entière
FORMAT: LEN (<chaîne>)

Rôle: Donne le nombre de caractères dans l'expression en chaîne. Les caractères non imprimés et les espaces vides sont comptés.

EXEMPLE de fonction LEN:

```
CC$ = "ORDINATEUR COMMODORE": PRINT LEN(CC$)

20
```

LET

TYPE: Instruction
FORMAT: [LET] <variable> = <expression>

Rôle: L'instruction LET peut servir à attribuer une valeur à une variable. Le mot LET est facultatif; de ce fait, les programmeurs expérimentés ne l'utilisent pas, car il est toujours sous-entendu et il occupe de la place précieuse en mémoire. Le signe égal (=) seul suffit dans l'attribution de la valeur d'une expression à un nom de variable.

EXEMPLES d'instruction LET:

```
10 LET D= 12                    (Équivaut à D = 12)
20 LET E$ = "ABC"
30 F$ = "MOTS"
40 SUM$ = E$ + F$               (SUM$ équivaut à ABCMOTS)
```

LIST

TYPE: Commande
FORMAT: LIST [[<première ligne>]—[<dernière ligne>]]

Rôle: La commande LIST permet d'observer des lignes du programme BASIC présentement dans la mémoire du Commodore 64. On peut ainsi utiliser le puissant éditeur d'écran de l'ordinateur pour corriger rapidement et facilement les programmes listés (LIST).

La commande LIST affiche, en totalité ou en partie, le programme présentement dans la mémoire du dispositif de sortie implicite. La liste est normalement dirigée vers l'écran; on peut utiliser l'instruction CMD pour faire passer la sortie à un dispositif extérieur comme une imprimante ou une unité de disque. La commande LIST peut apparaître dans un programme, mais le BASIC revient toujours au message READY du système après l'exécution d'une commande LIST.

Quand on fait apparaître la liste du programme sur l'écran, on peut ralentir le défilement du bas vers le haut de l'écran en appuyant sur la touche CTRL (commande). Pour suspendre la commande LIST, appuyer sur la touche RUN/STOP .

S'il n'est indiqué aucun numéro de ligne, le programme entier est listé. Si le numéro de première ligne est seul spécifié et s'il est suivi d'un tiret (—), le programme liste cette ligne et celles de numéros plus élevés. Si l'on spécifie seulement le numéro de dernière ligne, précédé d'un tiret, on liste alors toutes les lignes, du début du programme jusqu'à la ligne spécifiée. Si l'on spécifie les deux numéros, l'intervalle complet, y compris les numéros des lignes listées, est affiché.

EXEMPLES de commande LIST:

LIST	**(Liste le programme présentement en mémoire.)**
LIST 500	**(Liste la ligne 500 seulement.)**
LIST 150—	**(Liste toutes les lignes, de 150 à la fin.)**
LIST —1000	**(Liste toutes les lignes, du numéro le plus bas jusqu'à 1000.)**
LIST 150—1000	**(Liste les lignes 150 à 1000 incluse.)**

```
10 PRINT "VOICI LA LIGNE 10"
20 LIST (LIST est utilisée en mode de programme)
30 PRINT "VOICI LA LIGNE 30"
```

LOAD

TYPE: Commande
FORMAT: LOAD ["<nom de fichier>"] [,<dispositif>] [,<adresse>]

Rôle: La commande LOAD lit le contenu d'un fichier de programme sur cassette ou disque et le met en mémoire. On peut ainsi utiliser ou changer les informations chargées (LOAD). Le numéro de dispositif est facultatif, mais si on l'omet, l'ordinateur choisit implicitement le numéro 1 qui correspond au magnétocassette. L'unité de disques correspond normalement au dispositif 8. La commande LOAD ferme tous les fichiers ouverts; si on l'utilise en mode direct, elle exécute un effacement (CLR) avant de lire le programme. Si LOAD est utilisée dans un programme, celui-ci est exécuté (RUN). On peut donc utiliser la commande LOAD pour "enchaîner" plusieurs programmes. Aucune des variables n'est effacée pendant une opération en chaîne.

Si l'on utilise la concordance de configuration des noms de fichier, le premier fichier correspondant à la configuration est chargé. L'astérisque entre guillemets ("*") amène le chargement du premier nom de fichier dans le répertoire de disque. Si le nom de fichier utilisé n'existe pas ou ne correspond pas à un fichier de programme, on obtient le message d'erreur **?FILE NOT FOUND** (fichier non localisé).

Quand on charge (LOAD) des programmes à partir d'une cassette, on peut ignorer le <nom de fichier>; le fichier de programme suivant sur la cassette est alors lu. Le Commodore 64 fait passer l'écran à la couleur du cadre si l'on appuie sur la touche PLAY (lecture). Si le programme est localisé, l'écran passe à la couleur de fond et le message "FOUND" (localisé) apparaît. Si l'on appuie sur la touche C=, CTRL, ← ou SPACE BAR, le fichier est chargé. Le programme se charge à partir de la position de mémoire 2048, sauf si l'on utilise une <adresse> secondaire 1. Si l'on utilise l'adresse secondaire 1, le programme se charge à la position de mémoire où il avait été sauvegardé.

EXEMPLES de commande LOAD:

LOAD	(Lit le programme suivant de la cassette)
LOAD A$	(Utilise le nom dans A$ pour la recherche)
LOAD "*",8	(Charge le premier programme du disque)
LOAD "",1,1	(Cherche le premier programme sur cassette et le charge dans la position de mémoire d'où il venait)
LOAD "STAR TREK" PRESS PLAY ON TAPE FOUND STAR TREK LOADING READY.	(Charge un fichier de la cassette)
LOAD "LOISIRS",8 SEARCHING FOR LOISIRS LOADING READY	(Charge un fichier du disque)
LOAD "JEU UN",8,1 SEARCHING FOR JEU UN LOADING READY.	(Charge un fichier dans la position de mémoire spécifique d'où le programme avait été sauvegardé sur le disque)

LOG

TYPE: Fonction à point flottant
FORMAT: LOG (<numérique>)

Rôle: Donne le logarithme naturel (logarithme dans le système à base e) de l'argument. Si la valeur de l'argument est nulle ou négative, le message d'erreur **?ILLEGAL QUANTITY** (quantité interdite) apparaît.

EXEMPLES de fonction LOG:

25 PRINT LOG(45/7)
1.86075234

10 NUM = LOG(ARG) / LOG(10) (Calcule le LOG de l'argument dans le système à base 10)

MID$

TYPE: Fonction en chaîne
FORMAT: MID$ (<chaîne>, <numérique-1> [, <numérique-2>])

Rôle: La fonction MID$ donne une sous-chaîne tirée d'un argument de <chaîne> plus longue. La position de départ de la sous-chaîne est définie par l'argument <numérique-1> et sa longueur par l'argument <numérique-2>. Les deux arguments numériques peuvent être compris dans l'intervalle 0 à 255.

Si la valeur <numérique-1> est supérieure à la longueur de la <chaîne> ou si la valeur <numérique-2> est nulle, MID$ donne alors une valeur de chaîne vide. Si l'argument <numérique-2> est omis, l'ordinateur suppose alors que l'on utilise une longueur correspondant au reste de la chaîne. Si la chaîne d'origine a moins de caractères que l'argument <numérique-2>, de la position de début à la fin de l'argument de chaîne, le reste entier de la chaîne est alors utilisé.

EXEMPLE de fonction MID$:

```
10 A$ = "BON"
20 B$ = "JOUR SOIR APRÈS-MIDI"
30 PRINT A$ + MID$(B$, 6, 4)

BONSOIR
```

NEW

TYPE: Commande
FORMAT: NEW

Rôle: La commande NEW sert à annuler le programme présentement en mémoire et à effacer toutes les variables. Avant de taper un nouveau programme, utiliser NEW en mode direct pour effacer la mémoire. On peut aussi utiliser NEW dans un programme, mais il faut savoir qu'elle efface toutes les données qui étaient encore dans la mémoire de l'ordinateur. Cette situation peut être gênante si l'on essaie de mettre un programme au point.

> **ATTENTION:** Si l'on n'efface pas un ancien programme avant d'en taper un autre, les deux peuvent se trouver mélangés.

EXEMPLES de commande NEW:

NEW	**(Efface le programme et toutes les variables)**
10 NEW	**(Exécute une opération NEW et arrête (STOP) le programme)**

NEXT

TYPE: Instruction
FORMAT: NEXT [<compteur>] [,<compteur>] . . .

Rôle: L'instruction NEXT s'utilise avec FOR pour délimiter la fin d'une boucle FOR . . . NEXT. NEXT ne doit pas forcément être la dernière instruction de la boucle, mais elle correspond toujours à la dernière instruction qui y est exécutée. Le <compteur> correspond au nom de variable d'index de boucle utilisé avec FOR pour commencer la boucle. Une seule instruction NEXT peut arrêter plusieurs boucles emboîtées si elle est suivie par chaque nom de variable de <compteur> d'instruction FOR. Dans ce but, chaque nom doit apparaître dans l'ordre de la boucle emboîtée la plus intérieure, en progressant vers la boucle emboîtée la plus extérieure. Si l'on utilise une seule instruction NEXT pour faire progresser et arrêter plusieurs noms de variable, ceux-ci doivent être séparés par des virgules. On peut emboîter les boucles jusqu'à 9 niveaux. Si l'on omet les variables de compteur, on fait progresser le compteur rattaché à l'instruction FOR du niveau en cours (des boucles emboîtées).

Quand on atteint l'instruction NEXT, la valeur du compteur progresse de 1 ou d'une valeur de pas (STEP) facultative. Elle est ensuite vérifiée avec une valeur de fin pour déterminer si la boucle doit s'arrêter. Une boucle s'arrête quand on arrive à une instruction NEXT dont la valeur de compteur est supérieure à la valeur de fin.

EXEMPLES d'instruction NEXT:

10 FOR J=1 TO 5:FOR K = 10 TO 20:FOR N = 5 TO −5 STEP −1

20 NEXT N, K, J — **(Arrêt des boucles emboîtées)**

10 FOR L = 1 TO 100
20 FOR M = 1 TO 10
30 NEXT M
400 NEXT L — **(Remarquer que les boucles NE s'entrecroisent PAS)**

10 FOR A = 1 TO 10
20 FOR B = 1 TO 20
30 NEXT
40 NEXT — **(Remarquer qu'aucun nom de variable n'est nécessaire)**

NOT

TYPE: Opérateur logique
FORMAT: NOT <expression>

Rôle: L'opérateur logique NOT est le "complément" de la valeur de chaque bit dans son facteur simple. Il donne un résultat entier avec "complément à deux". Autrement dit, NOT signifie en fait: "Si ce n'est pas . . ."Avec un nombre à point flottant, les facteurs sont convertis en nombres entiers et les fractions sont éliminées. On peut aussi utiliser l'opérateur NOT dans une comparaison pour inverser la valeur vraie/fausse découlant d'un essai de relation; il inverse la signification de la comparaison. Dans le premier exemple ci-dessous, si le "complément à deux" de "AA" est égal à "BB" et si "BB" n'est pas (NOT) égal à "CC", l'expression est alors vraie.

EXEMPLES d'opérateur NOT:

```
10 IF NOT AA = BB AND NOT(BB = CC) THEN . . .
```

```
NN% = NOT 96: PRINT NN%
-97
```

> **REMARQUE:** Pour déterminer la valeur de NOT, utiliser l'expression X=(−(X+1)). (Le complément à deux d'une valeur entière est le complément du bit plus un.)

ON

TYPE: Instruction
FORMAT: ON <variable> GOTO / GOSUB <numéro de ligne> [,<numéro de ligne>] . . .

Rôle: L'instruction ON sert à aller (GOTO) à l'un de plusieurs numéros donnés de ligne, suivant la valeur d'une variable. La valeur des variables peut s'étendre de zéro jusqu'au nombre donné de lignes. Si la valeur n'est pas entière, la partie fractionnaire est omise. Par exemple, si la variable a la valeur 3, l'instruction ON va (GOTO) au troisième numéro de ligne dans la liste.

Si la valeur de la variable est négative, on obtient un message d'erreur **?ILLEGAL QUANTITY** (quantité interdite). Si la valeur est nulle ou supérieure au nombre d'éléments de la liste, le programme ne tient pas compte de l'instruction et continue avec celle qui suit l'instruction ON.

ON est en fait une variante peu utilisée de l'instruction IF . . . THEN . . . Dans une suite d'instructions IF, chacune d'elles envoie le programme à une ligne spécifique; une instruction ON peut remplacer une liste d'instructions IF. Dans le premier exemple ci-dessous, on peut remarquer qu'une instruction ON remplace 4 IF . . . THEN . . .

EXEMPLES d'instruction ON:

```
ON -(A=7)-2*(A=3)- 3*(A<3)-4*(A>7)GOTO 400,900,1000,100

ON X GOTO 100,130,180,220

ON X+3 GOSUB 9000,20,9000

100 ON NUM GOTO 150,300,320,390

500 ON SUM / 2 + 1 GOSUB 50, 80, 20
```

OPEN

TYPE: Instruction d'entrée/sortie
FORMAT: OPEN <numéro de fichier>, [<dispositif>] [,<adresse>] [,"<nom de fichier>[,<type>] [,<mode>]"]

Rôle: Cette instruction ouvre (OPEN) un canal pour l'entrée aussi bien que la sortie avec un dispositif périphérique. Il se peut cependant que l'on n'ait pas besoin de tous ces éléments avec chaque instruction OPEN. Dans certaines instructions OPEN, on n'utilise que deux codes:

1) NUMÉRO DE FICHIER LOGIQUE
2) NUMÉRO DE DISPOSITIF

Le <numéro de fichier> correspond au numéro de fichier logique qui rattache les instructions OPEN, CLOSE, CMD, GET#, INPUT# et PRINT# les unes aux autres et les associe au nom de fichier et à l'équipement utilisé. On peut attribuer au numéro de fichier logique tout nombre dans l'intervalle 1 à 255.

> **REMARQUE:** Les numéros de fichier supérieurs à 128 servent en fait à d'autres utilisations; il est donc bon de n'utiliser que des nombres inférieurs à 127 pour les numéros de fichier.

Chaque dispositif périphérique (imprimante, unité de disques, magnétocassette) possède un numéro auquel il répond. Le numéro de <dispositif> s'utilise avec l'instruction OPEN pour préciser le dispositif où se trouve le fichier de données. Les périphériques comme les magnétocassettes, les unités de disques et les imprimantes répondent aussi à plusieurs adresses secondaires. Ces adresses sont des codes qui indiquent l'opération à exécuter à chaque dispositif. Le numéro de fichier logique de dispositif s'utilise avec chaque instruction GET#, INPUT# et PRINT#.

Si le numéro de <dispositif> n'est pas indiqué, l'ordinateur suppose automatiquement que l'on désire communiquer avec le magnétocassette Datassette™, qui est le dispositif 1. On peut aussi négliger le nom de fichier mais, dans la suite du programme, on ne peut pas appeler le fichier s'il n'a pas reçu de nom. Quand on stocke des fichiers sur cassette, l'ordinateur suppose que l'<adresse> secondaire est zéro (0) si on ne l'indique pas (opération READ).

Une valeur d'adresse secondaire de un (1) ouvre (OPEN) les fichiers sur cassette pour l'écriture. Une valeur d'adresse secondaire de deux (2) amène l'écriture d'un repère de fin de bande quand le fichier est fermé par la suite. Après la fin des données, le repère de fin de bande évite la lecture accidentelle qui se traduit par le message d'erreur **?DEVICE NOT PRESENT** (pas de dispositif présent).

Avec les fichiers sur disque, les adresses secondaires 2 à 14 sont disponibles pour les fichiers de données, mais d'autres numéros ont des significations spéciales dans les commandes de système d'exploitation à disques (DOS). Une adresse secondaire est nécessaire quand on utilise une ou plusieurs unités de disques. (Pour plus de détails sur les commandes de système d'exploitation à disques, consulter le manuel de l'unité de disques.)

Le <nom de fichier> est une chaîne de 1 à 16 caractères, facultative pour les fichiers de cassette et d'imprimante. Si le <type> de fichier est omis, le type de fichier passe implicitement et automatiquement au fichier de *programme*, sauf si l'on indique le <mode>. Les fichiers *séquentiels* sont ouverts (OPEN) pour la lecture (<mode> =R), sauf si l'on précise qu'ils sont ouverts (OPEN) pour l'ecriture (<mode> =W). On peut utiliser un <type>de fichier pour ouvrir (OPEN) un fichier relatif *existant*. Avec un fichier relatif, utiliser <type> REL. Les fichiers relatifs et séquentiels ne s'utilisent qu'avec les disques.

Si l'on essaie d'accéder à un fichier avant qu'il soit ouvert, on obtient le message d'erreur **?FILE NOT OPEN** (fichier non ouvert). Si l'on essaie d'ouvrir pour la lecture un fichier qui n'existe pas, on obtient le message d'erreur **?FILE NOT FOUND** (fichier non localisé). Si l'on ouvre un fichier sur disque pour l'écriture et si le nom de fichier existe déjà, on obtient le message d'erreur de système d'exploitation à disques (DOS) **FILE EXISTS** (fichier existant). Il n'existe pas de vérification de ce genre pour les fichiers sur cassette; il faut donc s'assurer que la cassette est convenablement positionnée, sinon on risque d'écrire accidentellement sur des données précédemment sauvegardées. Si l'on ouvre un fichier qui l'est déjà, on obtient le message d'erreur BASIC **FILE OPEN** (fichier ouvert). (Pour plus de détails, consulter le manuel de l'imprimante.)

EXEMPLES d'instructions OPEN:

10 OPEN 2, 8, 4 "SORTIE-DISQUE, SEQ,W"	(Ouvre un fichier séquentiel sur disque)
10 OPEN 1, 1, 2, "ÉCRITURE SUR CASSETTE"	(Écrit fin de fichier à la fermeture)
10 OPEN 50, 0	(Entrée au clavier)
10 OPEN 12, 3	(Sortie sur écran)
10 OPEN 130, 4	(Sortie sur imprimante)
10 OPEN 1,1,0,"NOM"	(Lecture de cassette)
10 OPEN 1,1,1,"NOM"	(Écriture sur cassette)
10 OPEN 1,2,0, CHR$ (10)	(Ouverture d'un canal vers dispositif RS-232)
10 OPEN 1,4,0, "CHAINE"	(Envoi de caractères majuscules/graphiques à l'imprimante)
10 OPEN 1,4,7, "CHAINE"	(Envoi de caractères majuscules/minuscules à l'imprimante)
10 OPEN 1,5,7, "CHAINE"	(Envoi de caractères majuscules/minuscules à l'imprimante avec le dispositif 5)
10 OPEN 1,8,15, "COMMANDE"	(Envoi d'une commande au disque)

OR(OU)

TYPE: Opérateur logique
FORMAT: <facteur> OR <facteur>

Rôle: Les opérateurs de relation peuvent servir à prendre des décisions dans le déroulement d'un programme; les opérateurs logiques peuvent relier deux ou plusieurs relations et donner une valeur vraie ou fausse que l'on utilise ensuite dans une décision. Dans les calculs, le OR logique donne un résultat binaire de 1 si le bit correspondant de chacun ou des deux facteurs est 1. On obtient ainsi une valeur entière comme résultat, suivant les valeurs des facteurs. Utilisé dans des comparaisons, l'opérateur logique OR sert aussi à relier deux expressions en une seule expression mixte. Si l'une ou l'autre des expressions est vraie, la valeur de l'expression mixte est vraie (−1). Dans le premier exemple ci-dessous, si AA est égal à BB OU si XX est égal à 20, l'expression est vraie.

Dans l'utilisation des opérateurs logiques, les facteurs sont convertis en valeurs entières à 16 bits, avec signe et complément à deux dans l'intervalle de −32768 à +32767. Si les facteurs ne sont pas dans cet intervalle, il se produit un message d'erreur. Chaque bit du résultat est déterminé par les bits correspondants des deux facteurs.

EXEMPLES d'opérateur OR:

100 IF (AA = BB) OR (XX = 20) THEN . . .

230 KK% = 64 OR 32: PRINT KK%	**(On tape cette expression avec une valeur binaire de 1000000 pour 64 et de 100000 pour 32)**
96	**(L'ordinateur répond avec une valeur binaire de 1100000. 1100000=96.)**

PEEK

TYPE: Fonction entière
FORMAT: PEEK (<numérique>)

Rôle: Donne une valeur entière, dans l'intervalle 0 à 255, qui est lue d'une position de mémoire. L'expression <numérique> correspond à une position de mémoire qui doit se situer dans l'intervalle de 0 à 65535. Si ce n'est pas le cas, il se produit un message BASIC **?ILLEGAL QUANTITY** (quantité interdite).

EXEMPLES de fonction PEEK:

10 PRINT PEEK(53280) AND 15	**(Donne la valeur de la couleur du cadre de l'écran)**
5 A%=PEEK(45)+PEEK(46)*256	**(Donne l'adresse de la table de variables de BASIC)**

POKE

TYPE: Instruction
FORMAT: POKE <position>, <valeur>

Rôle: L'instruction POKE sert à écrire une valeur binaire d'un octet (8 bits) dans une position de mémoire donnée ou dans un registre d'entrée/sortie. La <position> est une expression arithmétique dont la valeur doit être comprise dans l'intervalle de 0 à 65535. La <valeur> est une expression qui peut se réduire à une valeur entière de 0 à 255. Si l'une ou l'autre de ces valeurs est en dehors de son intervalle respectif, on obtient un message d'erreur BASIC **?ILLEGAL QUANTITY** (quantité interdite).

L'instruction POKE et l'instruction PEEK (fonction incorporée pour examiner une position de mémoire) sont pratiques pour le stockage des données, le contrôle des affichages graphiques ou la création sonore, le chargement de sous-programmes en langage d'assemblage et la transmission d'arguments et de résultats avec les sous-programmes en langage d'assemblage. En outre, on peut examiner les paramètres de système d'exploitation avec les instructions PEEK ou les changer et les manipuler avec les instructions POKE. L'annexe G donne une topographie complète de mémoire des positions utiles.

EXEMPLES d'instruction POKE:

POKE 1024, 1	**(Met un "A" à la position 1, sur l'écran)**
POKE 2040, PTR	**(Met à jour le pointeur de données du caractère graphique programmable 0)**

10 POKE RED, 32
20 POKE 36879, 8
2050 POKE A, B

POS

TYPE: Fonction entière
FORMAT: POS (<fictif>)

Rôle: Indique la position présente du curseur qui est évidemment dans l'intervalle de 0 (caractère le plus à gauche) à 79 avec une ligne d'écran logique de 80 caractères. Le Commodore 64 ayant un écran de 40 colonnes, toute position de 40 à 79 se rapporte à la deuxième ligne d'écran. L'argument fictif est ignoré.

EXEMPLE de fonction POS:

1000 IF POS(0) > 38 THEN PRINT CHR$(13)

PRINT

TYPE: Instruction
FORMAT: PRINT [<variable>] [<,/;> <variable>] . . .

Rôle: L'instruction PRINT sert normalement à écrire des éléments de données sur l'écran. On peut toutefois utiliser l'instruction CMD pour rediriger la sortie vers un autre dispositif du système. Les <variables> dans la liste de sortie sont des expressions de tout type. S'il n'y a pas de liste de sortie, une ligne vide est imprimée. La ponctuation qui sépare les éléments de la liste de sortie détermine la position d'impression de chacun d'eux.

Pour la ponctuation, on peut utiliser des espaces vides, des virgules ou des points-virgules. La ligne d'écran logique de 80 caractères se divise en 8 zones d'impression de 10 espaces chacune. Dans la liste des expressions, une virgule amène l'impression de la valeur suivante au début de la zone suivante. Un point-virgule amène l'impression de la valeur suivante, immédiatement à la suite de la précédente, mais il existe deux exceptions à cette règle:

1) Éléments numériques suivis de l'addition d'un espace.

2) Nombres positifs précédés d'un espace.

L'utilisation d'espaces vides ou l'absence de ponctuation entre les constantes en chaîne ou les noms de variable équivaut à un point-virgule. Les espaces vides entre une chaîne et un élément numérique ou entre deux éléments numériques arrêtent la sortie, sans impression du deuxième élément.

En présence d'une virgule ou d'un point-virgule à la fin de la liste de sortie, l'instruction PRINT suivante commence l'impression sur la même ligne, avec les espaces correspondants. Si aucune ponctuation ne termine la liste, un retour du chariot et une avance de ligne sont envoyés à la fin des données. L'instruction PRINT suivante commence à la ligne suivante. Si les données imprimées dépassent 40 colonnes et si la sortie est dirigée vers l'écran, celle-ci continue à la ligne d'écran suivante.

En BASIC, aucune instruction n'offre autant de variété que PRINT. Elle s'accompagne d'un si grand nombre de symboles, de fonctions et de paramètres qu'on peut la considérer, dans le BASIC, comme un langage en soi, un langage spécialement prévu pour écrire sur l'écran.

EXEMPLES d'instruction PRINT:

1)

```
 5X = 5
10 PRINT -5*X, X-5, X+5, X ↑ 5

-25        0        10        3125
```

2)

```
 5 X=9
10 PRINT X;"ELEVE AU CARRE VAUT";X*X;"ET";
20 PRINT X "ELEVE AU CUBE VAUT" X ↑ 3

 9 ELEVE AU CARRE VAUT 81 ET 9 ELEVE AU CUBE VAUT 729
```

3)

```
 90 AA$="ALPHA":BB$="BRAVO": CC$="CHARLIE":DD$="DELTA":
    EE$="ECHO"
100 PRINT AA$BB$;CC$ DD$,EE$

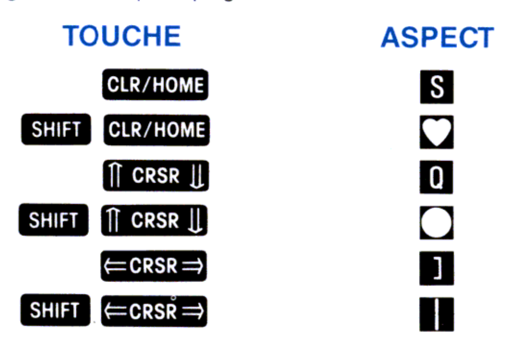
ALPHABRAVOCHARLIEDELTA        ECHO
```

Mode de guillemets

Quand on tape le guillemet (SHIFT 2), les commandes de curseur s'arrêtent puis se mettent à afficher les caractères inverses qui correspondent à la commande de curseur tapée. On peut ainsi programmer ces commandes parce que, après l'impression du texte entre guillemets, elles exécutent leurs fonctions. La touche INST/DEL est la seule commande de curseur qui n'est pas affectée par le mode de guillemets.

1. Déplacement du curseur

En mode de guillemets, on peut "programmer" les commandes de curseur suivantes:



Si l'on désire imprimer le mot SALUT en diagonale à partir du coin supérieur gauche de l'écran, taper:

PRINT " CLR /HOME **S** ⇑ CRSR ⇓ **A** ⇑ CRSR ⇓ **L** ⇑ CRSR ⇓ **U** ⇑ CRSR ⇓ **T"**
qui apparaît sous la forme:
PRINT " S **S** Q **A** Q **L** Q **U** Q **T"**

2. Caractères inverses

Si l'on appuie sur la touche CTRL et si l'on frappe 9 , R apparaît entre guillemets. Tous les caractères s'impriment alors en **vidéo inverse** (comme un négatif photographique). Pour interrompre les caractères inverses, appuyer sur CTRL et Ø ; on imprime un ⌴ . On peut aussi imprimer (PRINT) RETURN et (CHR$(13)). (Il suffit alors de terminer l'instruction PRINT sans point-virgule ni virgule.)

3. Commandes de couleur

Si l'on appuie sur la touche CTRL ou C= et sur l'une des 8 touches de couleur, un caractère inverse spécial apparaît entre les guillemets. Le changement de couleur se produit quand le caractère est imprimé (PRINT).

TOUCHE	COULEUR	ASPECT
CTRL 1	Noir	■
CTRL 2	Blanc	E
CTRL 3	Rouge	£
CTRL 4	Turquoise	◣
CTRL 5	Violet	
CTRL 6	Vert	↑
CTRL 7	Bleu	←
CTRL 8	Jaune	π
C= 1	Orange	♠
C= 2	Brun	
C= 3	Rouge clair	☒
C= 4	Gris 1	○
C= 5	Gris 2	♣
C= 6	Vert clair	
C= 7	Bleu clair	◆
C= 8	Gris 3	

Si l'on désire imprimer (PRINT) le mot BONJOUR en turquoise et le mot MONSIEUR en blanc, taper:

PRINT" CTRL 4 BONJOUR CTRL 2 MONSIEUR"

qui apparaît sous la forme:

PRINT" ◣ BONJOUR E MONSIEUR"

4. Mode d'insertion

Les espaces obtenus avec la touche INST/DEL possèdent certaines des caractéristiques du mode de guillemets. Les commandes de curseur et les commandes de couleur apparaissent en caractères inverses. La seule différence réside dans INST et DEL qui exécute sa fonction habituelle, même en mode de guillemets, et crée maintenant T, et INST, qui a donné un caractère spécial en mode de guillemets, insère normalement des espaces.

De ce fait, on peut créer une instruction PRINT avec des annulations (DEL) qui ne peuvent pas être imprimées (PRINT) en mode de guillemets. En voici un exemple:

10 PRINT"SALUT" INST/DEL SHIFT INST/DEL SHIFT INST/DEL INST/DEL INST/DEL **E"**

dont l'affichage est:

10 PRINT"SALUT T T **E"**

À l'exécution (RUN) de la ligne ci-dessus, le mot affiché est SALE car les deux dernières lettres sont annulées et remplacées par E.

> **AVERTISSEMENT:** Les annulations (DEL) marchent avec le listage (LIST) comme avec l'impression (PRINT). Il est donc difficile d'éditer une ligne avec ces caractères.

On termine le "mode d'insertion" en appuyant sur la touche RETURN (ou sur SHIFT et RETURN) ou quand on a tapé autant de caractères que l'on a inséré d'espaces.

5. Autres caractères spéciaux

On peut imprimer d'autres caractères pour des fonctions spéciales, mais ils ne sont pas facilement accessibles du clavier. Pour les mettre entre guillemets, on doit laisser des espaces vides à leur intention dans la ligne. Appuyer sur RETURN ou sur SHIFT et RETURN et revenir aux espaces avec les commandes de curseur. Appuyer ensuite sur CTRL et RVS/ON pour commencer la frappe des caractères inverses et taper sur les touches ci-dessous:

Fonction	Type	Aspect sur l'écran
SHIFT RETURN	SHIFT M	\
passage en minuscules	N	N
passage en majuscules	SHIFT N	/
invalidation des touches de permutation	H	H
validation des touches de permutation	I	I

SHIFT et RETURN s'utilisent avec le listage (LIST) comme avec l'impression (PRINT). L'édition est donc pratiquement impossible si l'on utilise ce caractère; le listage a aussi un aspect bizarre.

PRINT#

TYPE: Instruction d'entrée/sortie
FORMAT: PRINT#<numéro de fichier> [<variable>] [<,/;> <variable>] . . .

Rôle: L'instruction PRINT# sert à écrire des éléments de données dans un fichier logique. Elle doit utiliser le même numéro que pour l'ouverture (OPEN) du fichier. La sortie va au numéro de dispositif utilisé dans l'instruction OPEN. Les expressions de <variable> dans la liste de sortie peuvent être de n'importe quel type. Les caractères de ponctuation entre les éléments sont identiques à ceux de l'instruction PRINT; on peut les utiliser de façon identique. Les conséquences de la ponctuation sont différentes à deux points de vue.

Quand on utilise PRINT# avec des fichiers sur cassette, la virgule, au lieu de produire l'espacement par zone d'impression, a le même effet qu'un point-virgule. De ce fait, l'utilisation des espaces vides, des virgules, des points-virgules ou l'absence de ponctuation entre les éléments de données a le même effet sur l'espacement. Les éléments de données sont écrits en une suite continue de caractères. Les éléments numériques sont suivis d'un espace; s'ils sont positifs, ils sont aussi précédés d'un espace.

Si la liste se termine sans ponctuation, un retour de chariot et une avance de ligne sont écrits à la fin des données. Si une virgule ou un point-virgule termine la liste de sortie, le retour du chariot et l'avance de ligne sont supprimés. Quelle que soit la ponctuation, l'instruction PRINT# suivante commence la sortie à la position de caractère disponible suivante. L'avance de ligne sert d'arrêt quand on utilise l'instruction INPUT# et laisse une variable vide quand l'instruction INPUT# suivante est exécutée. On peut supprimer l'avance de ligne ou la rattraper comme l'indiquent les exemples ci-dessous.

Pour écrire facilement plus d'une variable dans un fichier sur cassette ou sur disque, on peut attribuer une variable en chaîne à CHR$(13) et utiliser la chaîne entre toutes les autres variables dans l'écriture du fichier.

EXEMPLES d'instruction PRINT#:

1)

```
10 OPEN 1,1,1, "FICHIER DE CASSETTE"
20 R$ = CHR$(13)
30 PRINT# 1,1;R$;2;R$;3;R$;4;R$;5
40 PRINT# 1,6
50 PRINT# 1,7
```

(En changeant CHR$(13) pour CHR$(44), on met une "," entre chaque variable. CHR$(59) mettrait un ";" entre chaque variable.)

2)

```
10 CO$=CHR$(44): CR$=CHR$(13)
20 PRINT#1,"AAA"CO$"BBB",
"CCC";"DDD";"EEE"CR$
"FFF"CR$;
30 INPUT#1, A$,BCDE$,F$
```

AAA,BBB CCCDDDEEE
(retour chariot)
FFF(retour chariot)

3)

```
 5 CR$=CHR$(13)
10 PRINT#2, "AAA";CR$;"BBB"
20 PRINT#2, "CCC";

30 INPUT#2, A$,B$,DUMMY$,C$
```

(10 espaces vides) AAA
BBB
(10 espaces vides)CCC

READ

TYPE: Instruction
FORMAT: READ <variable> [,<variable>] . . .

Rôle: L'instruction READ sert à remplir des noms de variable à partir de constantes dans les instructions DATA. Les données lues doivent correspondre aux types de variables spécifiés, sinon on obtient le message d'erreur BASIC **?SYNTAX ERROR** (erreur de syntaxe). *Les variables dans la liste d'entrée DATA doivent être séparées par des virgules.

Une seule instruction READ peut accéder à une ou plusieurs instructions DATA dans l'ordre (voir DATA). Plusieurs instructions READ peuvent accéder à la même instruction DATA. Si l'on exécute davantage d'instructions READ qu'il y a d'éléments dans les instructions DATA du programme, on obtient le message d'erreur BASIC **?OUT OF DATA** (fin des

données). Si le nombre spécifié de variables est inférieur au nombre d'éléments des instructions DATA, les instructions READ suivantes continuent à lire les éléments suivants de données. (Voir RESTORE.)

> *REMARQUE: Le message **?SYNTAX ERROR** (erreur de syntaxe) apparaît avec le numéro de ligne de l'instruction DATA et non avec l'instruction READ.

EXEMPLES d'instruction READ:

```
110 READ A,B,C$
120 DATA 1,2,BONJOUR

100 FOR X=1 TO 10: READ A(X):NEXT

200 DATA 3.08, 5.19, 3.12, 3.98, 4.24
210 DATA 5.08, 5.55, 4.00, 3.16, 3.37
```

(Remplit les éléments de tableau (ligne 1) dans l'ordre des constantes indiquées (ligne 5))

```
1 READ VILLE$,PROVINCE$,CODE POSTAL

5 DATA WINNIPEG,MANITOBA, M4B 3T4
```

REM

TYPE: Instruction
FORMAT: REM [<remarque>]

Rôle: L'instruction REM permet de mieux comprendre les programmes quand ils sont listés (LIST). Elle sert de rappel des objectifs de l'utilisateur quand il a écrit chaque section du programme. Par exemple, on peut vouloir se rappeler le rôle d'une variable ou d'autres indications utiles. La remarque (REM) peut comprendre texte, mot ou caractère, y compris les deux-points (:) ou les mots clés de BASIC.

L'instruction REM et les éléments qui la suivent sur le même numéro de ligne sont sautés par le BASIC. Les remarques sont cependant imprimées telles quelles quand on liste le programme. On peut se reporter à une instruction REM à l'aide d'une instruction GOTO ou GOSUB; l'exécution du programme continue avec la ligne de programme la plus élevée qui suit et qui contient des instructions d'exécution.

EXEMPLES d'instruction REM:

```
10 REM CALCUL DE LA VITESSE MOYENNE
20 FOR X=1 TO 20 :REM BOUCLE POUR VINGT VALEURS
30 SOM=SOM + VIT(X): NEXT
40 MOY=SOM/20
```

RESTORE

TYPE: Instruction
FORMAT: RESTORE

Rôle: Le BASIC maintient un pointeur interne sur les données DATA suivantes à lire (READ). On peut remettre ce pointeur à la première constante DATA dans un programme, avec l'instruction RESTORE. On peut utiliser cette instruction partout dans le programme pour commencer à relire (READ) les données.

EXEMPLES d'instruction RESTORE:

```
100 FOR X=1 TO 10: READ A(X): NEXT
200 RESTORE
300 FOR Y=1 TO 10: READ B(Y): NEXT

4000 DATA 3.08, 5.19, 3.12, 3.98, 4.24
4100 DATA 5.08, 5.55, 4.00, 3.16, 3.37
```

(Remplit les deux tableaux de données identiques)

```
10 DATA 1,2,3,4
20 DATA 5,6,7,8
30 FOR L=1 TO 8
40 READ A: PRINT A
50 NEXT
60 RESTORE
70 FOR L=1 TO 8
80 READ A: PRINT A
90 NEXT
```

RETURN

TYPE: Instruction
FORMAT: RETURN

Rôle: L'instruction RETURN sert à sortir d'un sous-programme appelé par une instruction GOSUB. RETURN reprend le reste du programme à l'instruction d'exécution qui suit GOSUB. Si l'on emboîte des sous-programmes, chaque instruction GOSUB doit être associée à une instruction RETURN au moins. Un sous-programme peut contenir un nombre varié d'instructions RETURN, mais la première rencontrée fait sortir du sous-programme.

EXEMPLE d'instruction RETURN:

```
10 PRINT "VOICI LE PROGRAMME"
20 GOSUB 1000
30 PRINT "LE PROGRAMME CONTINUE"
40 GOSUB 1000
50 PRINT "SUITE DU PROGRAMME"
60 END
1000 PRINT "VOICI LE SOUS-PROGRAMME":RETURN
```

RIGHT$

TYPE: Fonction en chaîne
FORMAT: RIGHT$ (<chaîne>, <numérique>)

Rôle: La fonction RIGHT$ retourne une sous-chaîne tirée de la partie la plus à droite de l'argument <chaîne>. La longueur de la sous-chaîne est définie par l'argument <numérique> qui peut être tout nombre entier de 0 à 255. Si la valeur de l'expression numérique est nulle, une chaîne vide ("") est retournée. Si la valeur donnée dans l'argument <numérique> est supérieure à la longueur de la <chaîne>, la chaîne complète est alors retournée.

EXEMPLE de fonction RIGHT$:

```
10 MSG$ = "ORDINATEURS COMMODORE"
20 PRINT RIGHT$(MSG$,9)
RUN

COMMODORE
```

RND

TYPE: Fonction à point flottant
FORMAT: RND (<numérique>)

Rôle: RND crée un nombre aléatoire à point flottant de 0.0 à 1.0. L'ordinateur crée une suite de nombres aléatoires en exécutant des calculs sur un *nombre de départ*. La fonction RND reçoit le nombre de départ à la mise sous tension du système. L'argument <numérique> est factice, hormis son signe (positif, nul ou négatif).

Si l'argument <numérique> est positif, la même suite "pseudo-aléatoire" de nombres est retournée, à partir d'une valeur de départ donnée. On obtient différentes suites de nombres à partir de nombres de départ différents, mais on peut répéter une suite en commençant avec le même nombre de départ. Une suite connue de nombres "aléatoires" est pratique dans la vérification des programmes.

Si l'on choisit l'argument <numérique> zéro, RND crée alors un nombre directement d'une horloge autonome (horloge du système). Les arguments négatifs amènent la fonction RND à recevoir un nouveau nombre de départ avec chaque appel de fonction.

EXEMPLES de fonction RND:

220 PRINT INT(RND(0)*50)	**(Retourne des nombres entiers aléatoires de 0 à 49)**
100 X=INT(RND(1)*6)+INT(RND(1)*6)+2	**(Simule 2 dés)**
100 X=INT(RND(1)*1000)+1	**(Nombres entiers aléatoires de 1 à 1000)**
100 X=INT(RND(1)*150)+100	**(Nombres aléatoires de 100 à 249)**
100 X=RND(1)*(U−L)+L	**(Nombres aléatoires entre limites supérieure et inférieure)**

RUN

TYPE: Commande
FORMAT: RUN [<numéro de ligne>]

Rôle: La commande de système RUN sert à commencer l'exécution du programme présentement en mémoire. Elle amène l'exécution d'une opération CLR implicite avant le début du programme. On peut éviter l'opération d'effacement (CLR) en utilisant CONT ou GOTO à la place de RUN pour reprendre un programme. Si un <numéro de ligne> est spécifié, le programme commence à cette ligne. Sinon, la commande RUN commence à la première ligne du programme. On peut aussi utiliser la commande RUN à l'intérieur d'un programme. Si le <numéro de ligne> spécifié n'existe pas, on obtient le message d'erreur BASIC **?UNDEF'D STATEMENT** (instruction non définie).

Un programme en cours d'exécution s'arrête et le BASIC revient en mode direct quand on arrive à une instruction END ou STOP, à la fin de la dernière ligne du programme ou en cas d'erreur BASIC pendant l'exécution.

EXEMPLES de commande RUN:

RUN	**(Commence à la première ligne du programme)**
RUN 500	**(Commence au numéro de ligne 500)**
RUN X	**(Commence à la ligne X ou donne une ERREUR D'INSTRUCTION NON DÉFINIE s'il n'y a pas de ligne X)**

SAVE

TYPE: Commande
FORMAT: SAVE ["<nom de fichier>"] [,<numéro de dispositif>] [,<adresse>]

Rôle: La commande SAVE sert à stocker le programme actuellement en mémoire sur un fichier de cassette ou de minidisque. Le programme sauvegardé n'est affecté par cette commande que pendant son exécution. Le programme reste dans la mémoire de l'ordinateur, même après la fin de la sauvegarde, jusqu'à ce qu'on l'y remplace en utilisant une autre commande. Le type de fichier est 'prg' (programme). Si l'on omet le <numéro de dispositif>, le C64 suppose alors automatiquement que l'on désire sauvegarder le programme sur le dispositif 1 (magnétocassette). Si le <numéro de dispositif> est <8>, le programme est alors écrit sur disque. On peut utiliser la commande SAVE dans les programmes; l'exécution continue avec l'instruction suivante quand la sauvegarde (SAVE) est terminée.

Les programmes sur cassette sont automatiquement stockés deux fois pour que le Commodore 64 puisse vérifier les erreurs lors du chargement (LOAD) de programme. Quand on charge des programmes sur cassette, le < nom de fichier > et < l'adresse > secondaire sont facultatifs. Si l'on fait suivre une commande SAVE d'un nom de programme entre guillemets ("") ou d'une variable en chaîne (—-$), le Commodore peut localiser plus facilement chaque programme. Si le nom de fichier est omis, on ne peut pas le charger (LOAD) en lui donnant un nom par la suite.

Une adresse secondaire 1 indique au KERNAL de charger (LOAD) la cassette plus tard, avec le programme présentement en mémoire plutôt qu'à la position normale 2048. Une adresse secondaire 2 met un marqueur de fin de bande à la suite du programme. Une adresse secondaire 3 combine les deux fonctions.

Si l'on sauvegarde des programmes sur disque, le < nom de fichier > doit être présent.

EXEMPLES de commande SAVE:

SAVE	**(Écrit sur cassette, sans nom)**
SAVE "ALPHA", 1	**(Mémorise sur cassette, avec nom de fichier "Alpha")**
SAVE "ALPHA", 1, 2	**(Mémorise "alpha" avec marqueur de fin de bande)**
SAVE "DISQUE JEU",8	**(Sauvegarde sur disque (dispositif 8))**
SAVE A$	**(Stocke sur cassette avec le nom A$)**
10 SAVE "SALUT"	**(Sauvegarde le programme et passe ensuite à la ligne de programme suivante)**
SAVE "MOI",1,3	**(Sauvegarde à la même position de mémoire et place un marqueur de fin de cassette)**

SGN

TYPE: Fonction entière
FORMAT: SGN (<numérique>)

Rôle: SGN donne une valeur entière, suivant le signe de l'argument <numérique>. Si l'argument est positif, on obtient 1; s'il est nul, on obtient 0; s'il est négatif, on obtient −1.

EXEMPLE de fonction SGN:

```
90 ON SGN(DV)+2 GOTO 100, 200, 300
(saute à 100 si DV=négatif, 200 si DV=0 et 300 si DV=positif)
```

SIN

TYPE: Fonction à point flottant
FORMAT: SIN (<numérique>)

Rôle: SIN donne le sinus de l'argument <numérique>, en radians. COS(x) équivaut à SIN(x+3.14159265/2).

EXEMPLE de fonction SIN:

```
235 AA = SIN(1.5): PRINT AA
.997494987
```

SPC

TYPE: Fonction en chaîne
FORMAT: SPC (<numérique>)

Rôle: La fonction SPC sert à assurer la mise en forme des données, comme sortie sur l'écran ou dans un fichier logique. Les espaces (SPC) donnés par l'argument <numérique> sont imprimés, à partir de la première position disponible. Pour l'écran ou les fichiers sur cassette, l'argument doit être dans l'intervalle de 0 à 255; il peut aller jusqu'à 254 pour les fichiers sur disque. Avec les fichiers d'imprimante, un retour de chariot et une avance de ligne automatique sont exécutés par l'imprimante si un espace (SPC) est imprimé à la dernière position de caractère d'une ligne. Aucun espace (SPC) n'est imprimé sur la ligne suivante.

EXEMPLE de fonction SPC:

```
10 PRINT "JUSTE"; "ICI";
20 PRINT SPC(5) "ET" SPC(14) "LA-BAS"
RUN

JUSTE ICI          ET                              LA-BAS
```

SQR

TYPE: Fonction à point flottant
FORMAT: SQR (<numérique>)

Rôle: SQR donne la racine carrée de l'argument <numérique>. L'argument ne doit pas être négatif sinon, on obtient le message d'erreur BASIC **?ILLEGAL QUANTITY** (quantité interdite).

EXEMPLE de fonction SQR:

```
FOR J = 2 TO 5: PRINT J*5, SQR(J * 5): NEXT

10 3.16227766
15 3.87298335
20 4.47213595
25 5
READY
```

STATUS

TYPE: Fonction entière
FORMAT: STATUS

Rôle: Donne un état (STATUS) d'exécution pour la dernière opération d'entrée/sortie conduite dans un fichier. La fonction STATUS peut se lire depuis tout dispositif périphérique. Le mot clé STATUS (ou ST) est un nom de variable défini par le système dans lequel le KERNAL place l'état (STATUS) des opérations d'entrée/sortie. Nous donnons ci-dessous une table des valeurs de code STATUS pour les opérations de fichier de cassette, d'imprimante, de disque et de RS-232:

Position de bit ST	Valeur numérique ST	Lecture cassette	Bus série lecture/écriture	Vérification + chargement
0	1		fin temps écriture	
1	2		fin temps lecture	
2	4	bloc court		bloc court
3	8	bloc long		bloc long
4	16	erreur lecture inaccessible		défaut de conformité
5	32	erreur total de contrôle		erreur total de contrôle
6	64	fin de fichier	fin ou identification	
7	−128	fin de bande	dispositif non présent	fin de bande

EXEMPLES de fonction STATUS:

```
10 OPEN 1, 4: OPEN 2, 8, 4, "FICHIER PRINCIPAL,SEQ,W"
20 GOSUB 100: REM VERIFICATION D'ETAT
30 INPUT#2, A$, B, C
40 IF STATUS AND 64 THEN 80: REM FIN DE FICHIER
50 GOSUB 100: REM VERIFICATION D'ETAT
60 PRINT#1, A$, B; C
70 GOTO 20
80 CLOSE1: CLOSE2
90 GOSUB 100: END
100 IF ST > 0 THEN 9000: REM ERREUR D'ENTREE/SORTIE FICHIER
110 RETURN
```

STEP

TYPE: Instruction
FORMAT: [STEP <expression>]

Rôle: Le mot clé STEP facultatif suit l'expression de <valeur de fin> dans une instruction FOR. Il définit une valeur de progression pour la variable de compteur de boucle. On peut utiliser toute valeur de progression. On obtient évidemment une boucle sans fin avec une valeur STEP nulle. Si le mot clé STEP est omis, la valeur de progression est +1. Dans une boucle FOR, quand on arrive à l'instruction NEXT, la progression STEP se produit. Le compteur est ensuite comparé à la valeur de fin pour déterminer si la boucle est terminée. (Pour plus de détails, voir l'instruction FOR.)

> **REMARQUE:** La valeur STEP NE PEUT PLUS changer quand elle est dans la boucle.

EXEMPLES d'instruction STEP:

25 FOR XX = 2 TO 20 STEP 2	**(La boucle se répète 10 fois)**
35 FOR ZZ = 0 TO −20 STEP −2	**(La boucle se répète 11 fois)**

STOP

TYPE: Instruction
FORMAT: STOP

Rôle: L'instruction STOP sert à interrompre l'exécution du programme en cours et à revenir en mode *direct*. Une pression sur la touche RUN/STOP du clavier a le même effet qu'une instruction STOP. Le message d'erreur BASIC **?BREAK IN LINE nnnnn** (interruption à la ligne nnnnn) apparaît sur l'écran, suivi de **READY** (prêt). "nnnnn" correspond au numéro de ligne où s'est produit l'arrêt. Les fichiers ouverts le restent et toutes les variables sont conservées et peuvent être examinées. On peut reprendre le programme à l'aide des instructions CONT ou GOTO.

EXEMPLES d'instruction STOP:

```
10 INPUT#1, AA, BB, CC
20 IF AA = BB AND BB = CC THEN STOP
30 STOP
```

(Si la variable AA est égale à −1 et si BB est égale à CC, on a alors:)
BREAK IN LINE 20
BREAK IN LINE 30 **(Pour les autres valeurs de données)**

STR$

TYPE: Fonction en chaîne
FORMAT: STR$ (<numérique>)

Rôle: STR$ donne la représentation en chaîne (STR) de la valeur numérique de l'argument. Quand la valeur STR$ est convertie à chaque variable représentée dans l'argument <numérique>, tout nombre indiqué est suivi d'un espace et, s'il est positif, il est également précédé d'un espace.

EXEMPLE de fonction STR$:

```
100 FLT = 1.5E4: ALPHA$ = STR$(FLT)
110 PRINT FLT, ALPHA$

15000      15000
```

SYS

TYPE: Instruction
FORMAT: SYS <position de mémoire>

Rôle: Cette instruction constitue le moyen le plus courant de mélanger un programme BASIC à un programme en langage machine. Ce dernier commence à la position indiquée dans l'instruction SYS. La commande de système SYS s'utilise en mode direct ou en mode de programme pour transférer la commande du microprocesseur à un programme en langage machine existant dans la mémoire. La position de mémoire donnée se trouve sous forme d'expression numérique, en un point quelconque de la mémoire vive (RAM) ou morte (ROM).

Quand on utilise l'instruction SYS, on doit terminer cette section de code en langage machine par une instruction RTS (retour du sous-programme) pour que, à la fin du programme en langage machine, l'exécution en BASIC reprenne avec l'instruction qui suit la commande SYS.

EXEMPLES d'instruction SYS:

SYS 64738	**(Saute au départ immédiat du système en mémoire morte (ROM))**
10 POKE 4400,96: SYS 4400	**(Passe à la position de code machine 4400 et revient immédiatement)**

TAB

TYPE: Fonction en chaîne
FORMAT: TAB (<numérique>)

Rôle: La fonction TAB déplace le curseur à une position SPC relative sur l'écran indiquée par l'argument <numérique>, à partir de la position la plus à gauche de la ligne en cours. La valeur de l'argument peut varier de 0 à 255. N'employer la fonction TAB qu'avec l'instruction PRINT, car elle n'a aucun effet si elle est utilisée avec PRINT# sur un fichier logique.

EXEMPLE de fonction TAB:

```
100 PRINT "NOM" TAB(25) "MONTANT": PRINT
110 INPUT#1, NOM$, MT$
120 PRINT NOM$ TAB(25) MT$

NOM                         MONTANT

G. LANDRY                   25
```

TAN

TYPE: Fonction à point flottant
FORMAT: TAN (<numérique>)

Rôle: Donne la tangente de l'expression <numérique> en radians. Si la fonction TAN donne un dépassement de capacité, le message d'erreur BASIC **?DIVISION BY ZERO** (division par zéro) apparaît.

EXEMPLE de fonction TAN:

```
10 XX = .785398163: YY = TAN(XX): PRINT YY
 1
```

TIME

TYPE: Fonction numérique
FORMAT: TI

Rôle: La fonction TI lit la minuterie d'intervalles. La valeur de cette "horloge" est fixée à zéro (initialisation) à la mise sous tension du système. La minuterie d'intervalles de 1/60 seconde est coupée pendant l'entrée/sortie sur cassette.

EXEMPLE de fonction TI:

```
10 PRINT TI/60 "SECONDES DEPUIS LA MISE SOUS TENSION"
```

TIME$

TYPE: Fonction en chaîne
FORMAT: TI$

Rôle: La minuterie TI$ marche comme une *horloge réelle* et lui ressemble tant que le système est sous tension. La minuterie d'intervalles de matériel est lue pour mettre à jour la valeur de TI$ qui donne une chaîne de temps (TI$) de six caractères en heures, minutes et secondes. On peut aussi attribuer à la minuterie TI$ un point de départ arbitraire, de la même manière que l'on règle une montre. La valeur de TI$ manque de précision après l'entrée/sortie de cassette.

EXEMPLE de fonction TI$:

```
1 TI$ = "000000": FOR J=1 TO 10000: NEXT: PRINT TI$

000011
```

USR

TYPE: Fonction à point flottant
FORMAT: USR (<numérique>)

Rôle: La fonction USR sert à sauter à un sous-programme en langage machine appelé par l'utilisateur et dont l'adresse de départ est indiquée par le contenu des positions de mémoire 785 et 786. On établit l'adresse de départ avant d'appeler la fonction USR en utilisant des instructions POKE pour fixer les positions 785 et 786. Si l'on n'utilise pas d'instructions POKE, les positions 785 et 786 donnent un message d'erreur **?ILLEGAL QUANTITY** (quantité interdite).

La valeur de l'argument <numérique> est stockée dans l'accumulateur à point flottant, à partir de la position 97, pour l'accès avec le code d'assembleur. La fonction USR donne la valeur qui se termine au point de retour du sous-programme au BASIC.

EXEMPLES de fonction USR:

```
10 B = T * SIN(Y)
20 C = USR (B/2)
30 D = USR (B/3)
```

VAL

TYPE: Fonction numérique
FORMAT: VAL (<chaîne>)

Rôle: Donne une valeur (VAL) numérique représentant les données dans l'argument de <chaîne>. Si le premier caractère non vide de la chaîne n'est pas un signe plus (+), un signe moins (−) ou un chiffre, la valeur (VAL) retournée est zéro. La conversion de chaîne est terminée à la localisation de la fin de la chaîne ou d'un caractère autre qu'un chiffre (sauf le point décimal ou le e exponentiel).

EXEMPLE de fonction VAL:

```
10 INPUT#1, NAM$, ZIP$
20 IF VAL(ZIP$) < 19400 OR VAL(ZIP$) > 96699
   THEN PRINT NAM$ TAB(25) "GRAND MONTREAL"
```

VERIFY

TYPE: Commande
FORMAT: VERIFY ["<nom de fichier>"] [,<dispositif>]

Rôle: En mode direct ou de programme, la commande VERIFY sert à comparer le contenu d'un fichier de programme BASIC sur cassette ou sur disque au programme présentement en mémoire. VERIFY s'utilise normalement après SAVE, pour s'assurer que le programme est convenablement mémorisé sur cassette ou sur disque.

Si le numéro de <dispositif> est omis, on suppose que le programme est sur le magnétocassette Datassette™ (dispositif 1). Avec les fichiers sur cassette, si le <nom de fichier> est omis, la comparaison se fait avec le programme suivant localisé sur la cassette. Avec les fichiers sur disque (dispositif 8), on doit avoir un nom de fichier. S'il se trouve des différences dans le texte du programme, le message d'erreur BASIC **?VERIFY ERROR** (vérifier erreur) est affiché.

Un nom de programme peut être indiqué entre guillemets (" ") ou en variable en chaîne. VERIFY sert aussi à positionner une bande de cassette après le dernier programme de façon à pouvoir en ajouter un nouveau sans écrire accidentellement par-dessus un autre programme.

EXEMPLES de commande VERIFY:

VERIFY	**(Vérifie le premier programme sur la cassette)**
PRESS PLAY ON TAPE **OK** **SEARCHING** **FOUND <NOM DE FICHIER>** **VERIFYING**	
9000 SAVE "MOI",8: **9010 VERIFY "MOI",8**	**(Cherche le programme dans le dispositif 8)**

WAIT

TYPE: Instruction
FORMAT: WAIT <position>, <masque 1>[, <masque 2>]

Rôle: L'instruction WAIT amène la suspension de l'exécution d'un programme jusqu'à ce qu'une adresse de mémoire donnée reconnaisse une configuration de bits spécifiée. On peut donc utiliser WAIT pour interrompre le programme jusqu'à ce qu'une condition extérieure se produise. Dans ce but, on contrôle l'état des bits dans les registres d'entrée/sortie. Avec WAIT, on peut utiliser des expressions numériques comme élément de données, mais elles sont converties en valeurs entières.

La plupart des programmeurs n'utilisent jamais cette instruction. Elle interrompt le programme jusqu'à ce que les bits d'une position spécifique de mémoire changent d'une manière particulière. Elle ne sert pratiquement et exclusivement qu'à certaines opérations d'entrée/sortie.

L'instruction WAIT prend la valeur dans la position de mémoire et exécute une opération logique ET avec la valeur du masque 1. S'il y a un masque 2 dans l'instruction, le résultat de la première opération est combiné avec le masque 2 dans une opération OU exclusif. Le masque 1 filtre donc les bits que l'on ne désire pas vérifier. Si le bit est 0 dans le masque 1, le bit correspondant du résultat est toujours 0. La valeur du masque 2 fait basculer les bits; on peut donc chercher un état d'arrêt ou un état de marche. Les bits vérifiés pour la valeur 0 doivent avoir un 1 à la position correspondante du masque 2.

Si les bits correspondants des facteurs <masque 1> et <masque 2> diffèrent, l'opération OU exclusif donne un résultat binaire de 1. Si les bits correspondants donnent le même résultat, le bit obtenu est 0. On peut introduire une pause de durée infinie avec l'instruction WAIT; dans ce cas, on peut utiliser les touches RUN/STOP et RESTORE pour reprendre. Appuyer sur la touche RUN/STOP puis sur RESTORE. Dans le premier exemple ci-dessous, on attend (WAIT) une pression sur une touche du magnétocassette pour continuer le programme. Dans le deuxième exemple, on attend (WAIT) qu'un caractère graphique programmable entre en collision avec l'arrière-plan de l'écran.

EXEMPLES d'instruction WAIT:

WAIT 1, 32, 32

WAIT 53273, 6, 6

WAIT 36868, 144, 16 — **(144 et 16 sont des masques. 144=10010000 en binaire et 16=10000 en binaire. L'instruction WAIT interrompt le programme jusqu'à ce que le bit 128 soit en marche ou jusqu'à ce que le bit 16 soit à l'arrêt)**

CLAVIER ET CARACTÉRISTIQUES DU COMMODORE 64

Le système d'exploitation possède une mémoire tampon de clavier à 10 caractères qui sert à conserver les frappes jusqu'à ce qu'elles puissent être traitées. Ce tampon ou queue retient les frappes dans l'ordre de leur arrivée; la première frappe de la queue est la première traitée. Par exemple, si l'on tape une deuxième touche avant le traitement de la première, la deuxième est stockée dans le tampon pendant que le traitement du premier caractère se poursuit. Quand le programme en a terminé avec le premier caractère, le tampon du clavier est examiné et la deuxième frappe est traitée. Sans ce tampon, l'introduction rapide au clavier amènerait de temps à autre la perte de caractères.

Le tampon du clavier permet donc de prendre de l'avance sur le système qui peut prévoir les réponses aux messages-guides d'entrée (INPUT) ou aux instructions GET. Quand on frappe les touches, leurs valeurs de caractère sont alignées en queue simple dans le tampon où elles attendent le traitement dans leur ordre d'entrée. Ce système de frappe peut parfois être gênant quand une frappe accidentelle amène un programme à extraire un caractère incorrect du tampon.

Dans des conditions normales, les frappes incorrectes ne présentent pas de problème, car on peut les corriger à l'aide des touches de curseur à gauche ⇐CRSR ou d'annulation INST/DEL et les retaper; les corrections sont traitées avant le retour suivant du chariot. Si l'on appuie toutefois sur la touche RETURN, il n'est plus possible de corriger, car tous les caractères du tampon jusqu'au retour du chariot y compris sont traités avant les corrections. On peut remédier à cette situation en utilisant une boucle pour vider le tampon de clavier avant la lecture d'une réponse prévue:

```
10 GET ERREUR$: IF ERREUR$ < >"" THEN 10: REM VIDAGE DU TAMPON DE
CLAVIER
```

En dehors des instructions GET et INPUT, on peut aussi lire le clavier avec l'instruction PEEK pour l'extraction à la position de mémoire 197 ($00C5) de la valeur entière de la touche présentement tapée. Si l'on n'appuie sur aucune touche pendant l'exécution de PEEK, la valeur 64 est retournée. L'annexe C donne les valeurs numériques du clavier, les symboles du clavier et les équivalents de caractère (CHR$). L'exemple suivant produit une boucle jusqu'à ce qu'on appuie sur une touche puis convertit la valeur entière en valeur de caractère.

```
10 AA = PEEK(197): IF AA = 64 THEN 10
20 BB$ = CHR$(AA)
```

Le clavier correspond à un jeu d'interrupteurs disposés dans une matrice de 8 colonnes et de 8 rangées. Le KERNAL, avec la microplaquette d'entrée/sortie CIA n° 1 (adaptateur d'interface complexe MOS 6526), explore les fermetures d'interrupteurs de touches de la matrice de clavier. Deux registres CIA servent à exécuter l'exploration: le registre 0 à la position 56320 ($DC00) pour les colonnes du clavier et le registre 1 à la position 56321 ($DC01) pour les rangées.

Les bits 0 à 7 de la position de mémoire 56320 correspondent aux colonnes 0 à 7. Les bits 0 à 7 de la position de mémoire 56321 correspondent aux rangées 0 à 7. En écrivant les valeurs des colonnes dans l'ordre et en lisant ensuite les valeurs des rangées, le KERNAL décode les fermetures d'interrupteurs dans la valeur CHR$ (N) de la touche pressée.

Avec huit colonnes et huit rangées, on dispose de 64 valeurs possibles. Toutefois, si l'on appuie d'abord sur la touche RVS , CTRL ou C= ou si l'on maintient la touche SHIFT et si l'on tape un deuxième caractère, on crée d'autres valeurs. En effet, le KERNAL décode séparément ces touches et "se souvient" si l'on a appuyé sur une des touches de commande. Le résultat de l'exploration du clavier est ensuite placé à la position 197.

On peut aussi écrire directement des caractères dans le tampon de clavier, aux positions 631 à 640, avec une instruction POKE. Ces caractères sont traités quand on utilise POKE pour établir un compte de caractères à la position 198. On peut utiliser ces conditions pour amener l'exécution automatique d'une série de commandes en mode direct en imprimant les instructions sur l'écran, en plaçant des retours de chariot dans le tampon et en fixant ensuite le compte de caractères. Dans l'exemple ci-dessous, le programme se liste (LIST) sur l'imprimante puis reprend l'exécution.

```
10 PRINT CHR$(147)"PRINT#1: CLOSE 1: GOTO 50"
20 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3
30 OPEN 1,4: CMD1: LIST
40 END
50 REM LE PROGRAMME REPREND ICI
```

ÉDITEUR D'ÉCRAN

L'ÉDITEUR D'ÉCRAN est un moyen pratique et puissant d'édition des textes de programme. Quand une section de programme est listée sur l'écran, on utilise les touches de curseur et les autres touches spéciales pour se déplacer sur l'écran et apporter les changements nécessaires. Après avoir apporté tous les changements désirés à un numéro de ligne spécifique de texte, on peut appuyer sur la touche RETURN en un point quelconque de la ligne pour que l'ÉDITEUR D'ÉCRAN lise la ligne d'écran logique complète de 80 caractères.

Le texte passe ensuite à l'interpréteur où il est symbolisé et stocké dans le programme. La ligne éditée remplace son ancienne version en mémoire. On peut créer une autre copie d'une ligne de texte en changeant simplement le numéro de ligne et en appuyant sur RETURN .

Si l'on utilise des abréviations de mots clés qui amènent une ligne de programme à dépasser 80 caractères, les caractères en trop sont perdus à l'édition de cette ligne, car l'éditeur ne lit que deux lignes physiques d'écran. Pour cette même raison, il n'est pas possible d'utiliser INPUT au-dessus de 80 caractères. À toutes fins pratiques, la longueur d'une ligne de texte BASIC est donc limitée aux 80 caractères affichés sur l'écran.

Dans certaines conditions, l'éditeur d'écran considère les touches de *commande de curseur* différemment de leur mode normal de manipulation. Si l'on place le curseur (CRSR) à la *droite* d'un *nombre impair* de guillemets (""), l'éditeur fonctionne en mode de guillemets.

En *mode de guillemets,* les caractères de données sont normalement introduits, mais le curseur n'est plus déplacé par ses commandes; on obtient à la place des caractères inverses qui correspondent à la commande de curseur introduite. Il en est de même avec les touches de commandes de couleur. On peut ainsi inclure des commandes de curseur et de couleur dans les éléments de données en chaîne des programmes. Cette caractéristique puissante est aussi très importante. En effet, quand le texte entre guillemets est imprimé sur l'écran, ce mode exécute automatiquement le positionnement du curseur et les fonctions de commande de couleur dans le cadre de la chaîne. Nous donnons ci-dessous un exemple de commande de curseur en chaînes.

Taper → **10 PRINT "A(R)(R)B(L)(L)(L)C(R)(R)D"**
:REM (R) = CRSR A DROITE, (L) = CRSR A GAUCHE

L'ordinateur imprime → AC BD

La touche DEL est la seule commande de curseur qui ne soit pas affectée par le mode de guillemets. De ce fait, s'il se produit une erreur pendant la frappe en mode de guillemets, on ne peut pas utiliser la touche ⇐CRSR pour reculer et taper par-dessus l'erreur; la touche INST elle-même donne un caractère en vidéo inverse. Il faut donc finir d'introduire la ligne et, après avoir appuyé sur la touche RETURN , on peut éditer normalement la ligne. Si l'on n'a plus besoin de commandes de curseur dans la chaîne, on peut aussi appuyer sur les touches RUN/STOP et RESTORE pour annuler le mode de guillemets. La table 2-2 donne les touches de commandes de curseur utilisables dans les chaînes.

Table 2-2. Caractères de commande de curseur en MODE DE GUILLEMETS

Touche de commande		Aspect
CRSR vers le haut	⇑ CRSR	●
CRSR vers le bas	CRSR ⇓	Q
CRSR vers la gauche	⇐CRSR	▌
CRSR vers la droite	CRSR⇒	]
CLR	CLR/	♥
HOME	/HOME	S
INST	INST/DEL	▐

Quand on n'est PAS en mode de guillemets, appuyer sur la touche SHIFT puis sur la touche INST pour faire passer les données à la droite du curseur et insérer un espace entre deux caractères pour y glisser des données. L'éditeur travaille ensuite en *MODE D'INSERTION* jusqu'à ce que la totalité de l'espace ouvert soit remplie.

Les commandes de curseur et les commandes de couleur apparaissent aussi en caractères inverses en mode d'insertion. Seule diffère la touche d'annulation/insertion INST/DEL. Au lieu de fonctionner normalement comme en mode de guillemets, la touche DEL crée maintenant un T inverse. La touche INST, qui crée un caractère inverse en mode de guillemets, insère normalement des espaces.

On peut donc créer une instruction PRINT, avec des annulations (DEL), chose impossible en mode de guillemets. On annule le mode d'insertion en appuyant sur les touches RETURN, SHIFT et RETURN ou RUN/STOP et RESTORE. On peut aussi annuler ce mode en remplissant tous les espaces insérés. Voici un exemple d'utilisation des caractères DEL dans des chaînes:

10 PRINT "SALUT" DEL INST INST DEL DEL E"
(Avec l'ordre de frappes ci-dessus, on obtient le message ci-dessous)
10 PRINT "SALE"

Si l'on exécute (RUN) cet exemple, on affiche le mot SALE, car les lettres UT ont été annulées avant l'impression du E. Un caractère anullé dans une chaîne marche avec le listage (LIST) comme avec l'impression (PRINT). On peut utiliser cette caractéristique pour "dissimuler" une ligne de texte en tout ou en partie. Il est cependant difficile, voire impossible, d'éditer une ligne avec ces caractères.

On peut imprimer certains autres caractères pour les fonctions spéciales, mais ils ne sont pas facilement accessibles depuis le clavier. Pour les mettre entre guillemets, on doit leur laisser des espaces vides dans la ligne, puis appuyer sur RETURN et revenir à l'édition de la ligne. Appuyer ensuite sur la touche de commande CTRL et sur la touche RVS/ON (vidéo inverse en fonction) pour commencer la frappe des caractères inverses. Taper sur les touches comme on l'indique ci-dessous:

Fonction de la touche	Touche introduite	Aspect
RETURN décalé	SHIFT M	
Passage aux majuscules/minuscules	N	N
Passage aux majuscules/graphiques	SHIFT N	

Si l'on appuie sur la touche SHIFT et sur RETURN, on amène un retour du chariot et une avance de ligne sur l'écran, mais on ne termine pas la chaîne. Cette opération marche avec le listage (LIST) comme avec l'impression (PRINT); l'édition est donc pratiquement impossible avec ce caractère. Si l'on fait passer la sortie à l'imprimante par l'intermédiaire de l'instruction CMD, le caractère "N" inverse fait passer l'imprimante en jeu de caractères majuscules/minuscules; SHIFT "N" fait passer l'imprimante au jeu de caractères majuscules/graphiques.

On peut inclure des caractères en vidéo inverse dans les chaînes en appuyant sur les touches de commande CTRL et vidéo inverse RVS pour faire apparaître un R inverse entre les guillemets. Tous les caractères sont alors imprimés en vidéo inverse (comme un négatif photographique). Pour terminer l'impression en vidéo inverse, appuyer sur les touches CTRL et RVS/OFF (arrêt vidéo inverse) pour imprimer un R inverse. On peut imprimer des données numériques en vidéo inverse en introduisant d'abord un code CHR$(18). Un code CHR$(146) ou un retour du chariot annule la sortie en vidéo inverse.

CHAPITRE **3**

PROGRAMMATION DES CARACTÈRES GRAPHIQUES AVEC LE COMMODORE 64

- Vue d'ensemble des caractères graphiques
- Position des caractères graphiques
- Mode de caractères normaux
- Caractères programmables
- Caractères graphiques en mode multicolore
- Mode étendu de couleur d'arrière-plan
- Caractères graphiques à topographie binaire
- Mode de topographie binaire multicolore
- Défilement uniforme
- Caractères graphiques programmables
- Autres caractéristiques des caractères graphiques
- Un autre aspect des caractères graphiques programmables

VUE D'ENSEMBLE DES CARACTÈRES GRAPHIQUES

Toutes les possibilités de caractères graphiques du Commodore 64 viennent de la microplaquette d'interface vidéo 6567 (puce VIC-II). Cette microplaquette donne différents modes de caractères graphiques, y compris un affichage de texte de 40 colonnes par 25 lignes, un affichage à haute définition de 320 par 200 points et les caractères graphiques programmables qui sont de petits objets mobiles simplifiant la préparation des jeux. On peut aussi mélanger plusieurs des modes de graphiques sur le même écran. Par exemple, on peut définir la moitié supérieure de l'écran en mode à haute résolution et la moitié inférieure en mode de texte. Les caractères graphiques programmables se combinent à pratiquement tous les autres! Nous verrons ces caractères plus tard. Examinons d'abord les autres modes de caractères graphiques.

La microplaquette VIC-II permet les modes d'affichage graphiques suivants:

A) MODES D'AFFICHAGE DE CARACTÈRES

1) Mode de caractères normaux
- a) Caractères en mémoire morte (ROM)
- b) Caractères programmables en mémoire vive (RAM)

2) Mode multicolore de caractères
- a) Caractères en mémoire morte (ROM)
- b) Caractères programmables en mémoire vive (RAM)

3) Mode étendu de couleurs d'arrière-plan
- a) Caractères en mémoire morte (ROM)
- b) Caractères programmables en mémoire vive (RAM)

B) MODES DE TOPOGRAPHIE BINAIRE

1) Mode de topographie binaire normal
2) Mode de topographie binaire multicolore

C) CARACTÈRES GRAPHIQUES PROGRAMMABLES

1) Caractères graphiques programmables normaux
2) Caractères graphiques programmables multicolores

POSITIONS DES CARACTÈRES GRAPHIQUES

Commençons par quelques généralités. L'écran du Commodore 64 comprend 1000 positions possibles. L'écran commence normalement à la position 1024 ($0400 en notation hexadécimale) et va jusqu'à la position 2023. Chacune de ces positions a une largeur de 8 bits. Elle peut donc recevoir un nombre entier de 0 à 255. Un groupe de 1000 positions dit **MÉMOIRE COULEUR** ou **MÉMOIRE VIVE COULEUR (RAM)** est raccordé à la mémoire d'écran. Cette mémoire commence à la position 55296 ($D800 en hexadécimal) et va jusqu'à 56295. Chacune des positions de la mémoire vive couleur a une largeur de 4 bits; elle peut recevoir un nombre entier de 0 à 15. Cette disposition concorde avec les 16 couleurs possibles permises par le Commodore 64.

D'autre part, on peut afficher 256 caractères différents à tout instant. Pour l'affichage normal d'écran, chacune des 1000 positions de la mémoire d'écran contient un numéro de code qui indique à la microplaquette VIC-II le caractère à afficher à la position d'écran donnée.

Les différents modes de caractères graphiques sont choisis par les 47 registres de **COMMANDE** de la microplaquette VIC-II. On peut commander la majeure partie des fonctions de caractères graphiques en insérant la valeur correcte avec une instruction POKE dans l'un des registres. La microplaquette VIC-II commence à la position 53248 ($D000 en hexadécimal) et va jusqu'à 53294 ($D02E en hexadécimal).

CHOIX DES BLOCS VIDÉO

La microplaquette VIC-II peut accéder à 16 K de mémoire à la fois. Le Commodore ayant 64 K de mémoire, on peut désirer que la VIC-II puisse accéder à la totalité de cette mémoire. Dans ce but, il existe 4 **BLOCS** possibles de 16 K de mémoire. Il suffit de disposer d'un moyen de contrôler le bloc de 16 K auquel la VIC-II peut accéder. La microplaquette peut ainsi "voir" la totalité des 64 K de mémoire. Les bits **DE SÉLECTION DE BLOC** qui permettent l'accès aux différentes sections de mémoire se trouvent dans **LA MICROPLAQUETTE nº 2 D'ADAPTATEUR D'INTERFACE COMPLEXE 6526 (CIA nº 2).** Les instructions BASIC POKE et PEEK (ou la version correspondante en langage machine) servent à choisir un bloc en commandant les bits 0 et 1 de l'accès A de la CIA nº2 (position 56576 ou $DD00 en hexadécimal). Ces 2 bits doivent être fixés en *sorties* en réglant les bits 0 et 1 de la position 56578 ($DD02 en hexadécimal) si l'on veut changer les blocs. Le programme suivant en donne un exemple:

POKE 56578, PEEK(56578)OR 3: REM LES BITS 0 ET 1 DOIVENT ÊTRE FIXÉS EN SORTIES

POKE 56576,(PEEK(56576)AND 252)OR A:REM CHANGEMENT DE BLOCS

"A" doit avoir l'une des valeurs suivantes:

VALEUR DE A	BITS	BLOC	POSITION DE DÉPART	INTERVALLE DE LA VIC-II
0	00	3	49152	($C000-$FFFF)*
1	01	2	32768	($8000-$BFFF)
2	10	1	16384	($4000-$7FFF)*
3	11	0	0	($0000-$3FFF) (VALEUR IMPLICITE)

Ce concept de blocs de 16 K fait partie intégrante des possibilités de la microplaquette VIC-II. On doit toujours connaître la section pointée par la VIC-II, car elle affecte la provenance des configurations de données de caractères, la position de l'écran, la provenance des caractères graphiques programmables, etc. Quand on met le Commodore 64 en marche, les bits 0 et 1 de la position 56576 sont automatiquement fixés au bloc 0 ($0000-$3FFF) pour toutes les informations d'affichage.

***REMARQUE:** Le jeu de caractères du Commodore 64 n'est pas disponible avec la microplaquette VIC-II aux blocs 1 et 3. (voir la section sur la mémoire de caractères.)

MÉMOIRE D'ÉCRAN

On peut facilement changer la position de la mémoire d'écran par une instruction POKE dans le registre de commande 53272 ($D018 en hexadécimal). Toutefois, ce registre sert aussi à commander le jeu de caractères utilisé; il faut donc faire attention à éviter de perturber cette partie du registre de commande. Les **4 bits supérieurs** commandent la position de la mémoire d'écran. Pour déplacer l'écran, utiliser l'instruction suivante:

POKE53272,(PEEK(53272)AND15)ORA

dans laquelle "A" a l'une des valeurs suivantes:

A	Bits	Position*	
		Décimal	Hexadécimal
0	0000XXXX	0	$0000
16	0001XXXX	1024	$0400 (IMPLICITE)
32	0010XXXX	2048	$0800
48	0011XXXX	3072	$0C00
64	0100XXXX	4096	$1000
80	0101XXXX	5120	$1400
96	0110XXXX	6144	$1800
112	0111XXXX	7168	$1C00
128	1000XXXX	8192	$2000
144	1001XXXX	9216	$2400
160	1010XXXX	10240	$2800
176	1011XXXX	11264	$2C00
192	1100XXXX	12288	$3000
208	1101XXXX	13312	$3400
224	1110XXXX	14336	$3800
240	1111XXXX	15360	$3C00

*Ne pas oublier d'ajouter l'adresse de bloc de la microplaquette VIC-II.
On doit aussi indiquer à l'éditeur d'écran de KERNAL la position de l'écran de la façon suivante: POKE 648, page (page = adresse/256, soit 1024/256 = 4, donc POKE 648,4).

MÉMOIRE DE COULEURS

On NE PEUT PAS déplacer la mémoire de couleur; elle est toujours aux positions 55296 ($D800) à 56295 ($DBE7). On utilise différemment la mémoire d'écran (1000 positions commençant à 1024) et la mémoire de couleurs dans les divers modes de caractères graphiques. Une image, créée dans un mode, a souvent un aspect totalement différent si on l'affiche dans un autre mode de graphiques.

MÉMOIRE DE CARACTÈRES

La provenance exacte des informations sur les caractères pour la VIC-II est très importante en programmation graphique. La microplaquette reçoit normalement les formes des caractères que l'on veut afficher de la mémoire morte (ROM) du **GÉNÉRATEUR DE CARACTÈRE.** Cette microplaquette stocke les *configurations* qui forment les divers nom-

bres, lettres, symboles de ponctuation et autres éléments indiqués au clavier. Le Commodore 64 présente l'avantage de pouvoir utiliser des configurations placées en mémoire vive (RAM). Ces configurations en mémoire vive sont créées par l'utilisateur; on peut donc disposer d'un jeu pratiquement infini de symboles pour les jeux, les applications de gestion, etc.

Un jeu normal contient 256 caractères dont chacun est défini par 8 octets de données. Chaque caractère prend jusqu'à 8 octets; de ce fait, un jeu complet occupe donc 256*8=2 K-octets de mémoire. La microplaquette VIC-II accède à 16 K de mémoire à la fois; de ce fait, il existe 8 positions possibles pour un jeu complet de caractères. On n'est évidemment pas tenu d'utiliser un jeu entier de caractères, mais il doit quand même commencer à l'une des 8 positions possibles de départ.

La position de la mémoire de caractères est commandée par 3 bits du registre de commande de la VIC-II situés à 53272 ($D018 en hexadécimal). Les bits 3, 2 et 1 commandent la position du jeu de caractères dans des blocs de 2 K-octets. Le bit 0 est sauté. Il faut se rappeler que ce même registre détermine aussi la position de la mémoire d'écran; il faut donc éviter de déranger les bits de mémoire d'écran. Pour changer la position de la mémoire de caractères, on peut utiliser l'instruction BASIC suivante:

POKE 53272,(PEEK(53272)AND240)OR A

dans laquelle "A" est l'une des valeurs suivantes:

VALEUR DE A	BITS	POSTITON DE LA MÉMOIRE DE CARACTÈRES*	
		DÉCIMAL	HEXADÉCIMAL
0	XXXX000X	0	$0000-$07FF
2	XXXX001X	2048	$0800-$0FFF
4	XXXX010X	4096	$1000-$17FF IMAGE ROM DANS BLOCS 0 et 2 (implicite)
6	XXXX011X	6144	$1800-$1FFF IMAGE ROM DANS BLOCS 0 et 2
8	XXXX100X	8192	$2000-$27FF
10	XXXX101X	10240	$2800-$2FFF
12	XXXX110X	12288	$3000-$37FF
14	XXXX111X	14336	$3800-$3FFF

*Ne pas oublier **d'ajouter l'adresse de bloc.**

Dans la table ci-dessus, **L'IMAGE ROM** se rapporte à la mémoire morte (ROM) du générateur de caractère. Elle apparaît à la place de la mémoire vive RAM aux positions ci-dessus, dans le bloc 0. Elle apparaît aussi dans la mémoire vive RAM correspondante aux positions 36864-40959 ($9000-$9FFF) dans le bloc 2. La VIC-II ne peut accéder qu'à 16 K de mémoire à la fois; les configurations de caractères de mémoire morte ROM n'apparaissent que dans le bloc 16 K de mémoire observé par la VIC-II. De ce fait, le système a été conçu pour que la VIC-II "pense" que les caractères de mémoire morte sont aux positions 4096-8191 ($1000-$1FFF) quand les données sont dans le bloc 0 et aux positions 36864-40959 ($9000-$9FFF) quand les données sont dans le bloc 2, même si la mémoire ROM de caractères est en fait à la position 53248-57343 ($D000-$DFFF). Ce processus d'image ne s'applique qu'aux données de caractères vues par la microplaquette VIC-II. On peut l'utiliser avec les programmes, d'autres données, etc. comme toute autre mémoire vive RAM.

REMARQUE: Si ces images de mémoire morte ROM gênent les graphiques de l'utilisateur, fixer alors les **bits de sélection de bloc** à l'un des blocs, *sans* les images (bloc 1 ou 3). Les configurations de mémoire morte ROM n'y sont pas.

La position et le contenu du jeu de caractères en mémoire morte ROM sont les suivants:

BLOC	ADRESSE		IMAGE VIC-II	CONTENUS
	DÉCIMAL	HEXADÉCIMAL		
0	53248	D000-D1FF	1000-11FF	Caractères majuscules
	53760	D200-D3FF	1200-13FF	Caractères graphiques
	54272	D400-D5FF	1400-15FF	Caractères majuscules inverses
	54784	D600-D7FF	1600-17FF	Caractères graphiques inverses
1	55296	D800-D9FF	1800-19FF	Caractères minuscules
	55808	DA00-DBFF	1A00-1BFF	Caractères majuscules et graphiques
	56320	DC00-DDFF	1C00-1DFF	Caractères miniscules inverses
	56832	DE00-DFFF	1E00-1FFF	Caractères graphiques et majuscules inverses

Le lecteur attentif aura probablement remarqué que les positions occupées par la mémoire morte ROM de caractères sont identiques à celles des registres de commande de microplaquette VIC-II. Cette situation est possible, car ils n'occupent pas les mêmes positions en même temps. Quand la VIC-II doit accéder aux données de caractères, la mémoire morte ROM est en fonction. Elle devient une image de bloc de mémoire de 16 K observé par la VIC-II. Dans les autres cas, la zone est occupée par les registres de commande d'entrée/sortie et la mémoire morte ROM de caractères n'est accessible que par la VIC-II.

On peut cependant avoir besoin d'accéder à la mémoire morte ROM de caractères si l'on compte utiliser des caractères programmables et si l'on veut copier une partie de cette mémoire pour certaines définitions de caractères. Dans ce cas, on doit couper le registre d'entrée/sortie, mettre la mémoire morte ROM de caractères en fonction et procéder à la copie. Quand on a terminé, on doit remettre les registres d'entrée/sortie en fonction. Pendant la copie (quand l'entrée/sortie est coupée), il ne peut pas se produire d'interruption. En effet, les registres d'entrée/sortie sont nécessaires pour le traitement des interruptions. Si l'on exécute une interruption par mégarde, il se produit d'étranges réactions. Ne pas lire le clavier pendant le processus de copie. Pour arrêter le clavier et les autres interruptions normales qui se produisent avec le Commodore 64, utiliser l'instruction POKE suivante:

POKE 56334,PEEK(56334)AND254 (COUPE LES INTERRUPTIONS)

Quand on n'extrait plus de caractères de la mémoire morte ROM de caractères et que l'on est prêt à poursuivre le programme, on doit revenir à l'exploration du clavier avec l'instruction POKE suivante:

POKE 56334,PEEK(56334)OR1 (REMISE EN FONCTION DES INTERRUPTIONS)

L'instruction POKE suivante coupe l'entrée/sortie et met la mémoire morte de caractères en fonction:

POKE 1,PEEK(1)AND251

La mémoire morte ROM de caractères est maintenant aux positions 53248-57343 ($D000-$DFFF).

Pour ramener l'entrée/sortie dans $D000 pour l'utilisation normale, envoyer l'instruction POKE suivante:

POKE 1,PEEK(1)OR 4

MODE DE CARACTÈRES NORMAUX

À la première mise en marche, le Commodore 64 est en mode de caractères normaux. On utilise généralement ce mode pour la programmation.

On peut extraire des caractères de la mémoire morte (ROM) ou vive (RAM), mais ils viennent généralement de la mémoire ROM. Si l'on a besoin de caractères graphiques spéciaux, il suffit de définir les nouvelles formes de caractères en mémoire vive RAM et d'indiquer à la microplaquette VIC-II de prendre ses informations de caractères dans cette mémoire au lieu de la mémoire morte ROM de caractères. La section suivante explique ce sujet plus en détail.

Pour afficher les caractères en couleur sur l'écran, la microplaquette VIC-II accède à la mémoire d'écran pour déterminer le code de caractère pour la position sur l'écran. En même temps, elle accède à la mémoire de couleurs pour déterminer la couleur désirée d'affichage du caractère. Le code de caractère est traduit par la VIC-II en adresse de départ du bloc de 8 octets contenant la configuration de caractère. Le bloc de 8 octets se trouve dans la mémoire de caractères.

La traduction n'est pas très compliquée; plusieurs éléments sont réunis pour créer l'adresse désirée. Le code de caractère utilisé pour écrire les instructions POKE en mémoire d'écran est d'abord multiplié par 8. On ajoute ensuite le début de la mémoire de caractères (voir la section ''MÉMOIRE DE CARACTÈRES''). On prend ensuite en considération les bits de sélection de bloc en les ajoutant dans l'adresse de base (voir section 'SÉLECTION DES BLOCS VIDÉO''). Nous donnons ci-dessous une formule simple qui montre cette traduction:

ADRESSE DE CARACTÈRE = CODE D'ÉCRAN * 8 + (JEU DE CARACTÈRES*2048) + (BLOC*16384)

DÉFINITIONS DES CARACTÈRES

Chaque caractère est formé d'une grille de 8 par 8 points dans laquelle chaque point peut être en ou hors fonction. Les images de caractères du Commodore 64 sont stockées dans la microplaquette de mémoire morte ROM du générateur de caractère. Chacun des caractères est stocké sous forme d'un jeu de 8 octets; chaque octet représente la configuration de points d'une rangée du caractère et chaque bit correspond à un point. Un bit zéro correspond à un point hors fonction et un bit 1 à un point en fonction.

La mémoire de caractères en mémoire morte ROM commence à la position 53248 (quand l'entrée/sortie est coupée). Les 8 premiers octets de la position 53248 ($D000) à 53255 ($D007) contiennent la configuration du signe @ qui a une valeur de code de caractère de zéro dans la mémoire d'écran. Les 8 octets suivants, de la position 53256 ($D008) à 53263 ($D00F), contiennent les informations de formation de la lettre A.

IMAGE	BINAIRE	PEEK
	00011000	24
	00111100	60
	01100110	102
****	01111110	126
** **	01100110	102
** **	01100110	102
** **	01100110	102
	00000000	0

Chaque jeu complet de caractères prend jusqu'à 2 K-octets (2048 bits) de mémoire pour 256 caractères, à raison de 8 octets chacun. Il y a deux jeux de caractères dont un pour les majuscules et les graphiques et un autre pour les majuscules et minuscules; la mémoire morte ROM de générateur de caractère prend donc un total de 4 K positions.

CARACTÈRES PROGRAMMABLES

Les caractères étant stockés en mémoire morte ROM, on pourrait penser qu'il n'est pas possible de les changer pour obtenir des caractères "sur mesure". Toutefois, la position de mémoire qui indique à la VIC-II l'emplacement des caractères est un registre programmable que l'on peut changer pour pointer vers de nombreuses sections de la mémoire. En changeant l'indicateur de mémoire de caractères pour qu'il pointe vers la mémoire vive RAM, on peut programmer le jeu de caractères en fonction des besoins.

Si l'on veut placer le jeu de caractères en mémoire vive RAM, il faut se rappeler quelques éléments **TRÈS IMPORTANTS** quand on décide de programmer soi-même des jeux de caractères. D'autre part, il ne faut pas oublier les deux autres points suivants dans la création de caractères spéciaux:

1) Ce processus oblige à faire un choix. En général, si l'on utilise un jeu de caractères spéciaux en indiquant à la VIC-II d'extraire les informations de caractères de la zone préparée en mémoire vive RAM, on ne peut alors pas disposer des caractères normaux du Commodore 64. Pour remédier à cet inconvénient, on doit copier les lettres, nombres ou caractères graphiques normaux du Commodore 64 que l'on veut utiliser dans la mémoire de caractères de l'utilisateur, en mémoire vive RAM. On peut choisir et extraire les caractères que l'on désire, sans qu'ils soient dans un ordre déterminé.

2) Le jeu de caractères de l'utilisateur enlève de la place de mémoire pour le programme BASIC. Néanmoins, avec 38 K-octets disponibles pour un programme BASIC, on ne doit avoir aucun problème avec la plupart des applications.

AVERTISSEMENT: Faire attention de ne pas "écraser" le jeu de caractères avec le programme BASIC qui utilise aussi de la mémoire vive RAM.

Dans le Commodore 64, il existe deux positions de départ du jeu de caractères **qu'il ne faut pas utiliser en BASIC: ce sont la position 0 et la position 2048.** On ne doit pas utiliser la première parce que le système stocke des données importantes à la page 0. On ne peut pas utiliser la deuxième parce qu'elle correspond au point de départ du programme BASIC. On peut cependant faire commencer le jeu de caractères spéciaux à 6 autres positions.

La position 12288 ($3000 en hexadécimal) est le meilleur point de départ du jeu de caractères en BASIC. Dans ce but, inscrire (POKE) 12 dans les 4 bits bas de la position 53272. Essayer ensuite d'écrire (POKE) la ligne suivante:

```
POKE 53272,(PEEK(53272)AND240)+12
```

Toutes les lettres de l'écran se transforment immédiatement en informations parasites, car il n'y a pas de caractères établis pour le moment à la position 12288; il n'y a que des octets au hasard. Ramener le Commodore 64 à la normale en appuyant sur la touche **RUN/STOP** puis sur la touche **RESTORE** .

Passons maintenant à la création des caractères graphiques. Pour protéger le jeu de caractères du BASIC, on doit réduire la quantité de mémoire que le BASIC "pense" avoir. La quantité de mémoire de l'ordinateur reste la même . . . on a juste demandé au BASIC de ne pas en utiliser une partie. Taper:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

Le nombre affiché correspond à la place de mémoire inutilisée. Taper maintenant la ligne suivante:

```
POKE 52,48:POKE56,48:CLR
```

Taper ensuite:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

Le BASIC 'pense'' maintenant avoir moins de mémoire. Dans la mémoire que l'on vient de reprendre au BASIC, nous pouvons maintenant mettre le jeu de caractères, à l'abri des réactions du BASIC.

Il faut ensuite mettre les caractères en mémoire vive RAM. Au début, des données aléatoires commencent à 12288 ($3000 en hexadécimal). On doit mettre les configurations de caractères en mémoire vive RAM (de la même manière que ceux mis en mémoire morte ROM) pour leur utilisation par la VIC-II.

Le programme suivant fait passer 64 caractères de la mémoire morte ROM dans la mémoire vive RAM de jeu de caractères:

```
5 PRINTCHR$(142)                              :PASSE EN MAJUSCULES
10 POKE52,48:POKE56,48:CLR                    :REM GARDE DE LA MEMOIRE
POUR LES CARACTERES
20 POKE56334,PEEK(56334)AND254                :REM COUPE EXPLORATION DE
CLAVIER ET INTERROMPT MINUTERIE
30 POKE1, PEEK(1)AND251                       :REM PASSE EN CARACTERES
40 FORI=0TO511:POKEI+12288,PEEK(I+53248)      :NEXT
50 POKE1,PEEK(1)OR4                           :REM PASSE EN ENTREE/SORTIE
60 POKE56334,PEEK(56334)OR1                   :REM REMET EN MARCHE
EXPLORATION DU CLAVIER ET INTERROMPT MINUTERIE
70 END
```

À la position 53272, écrire (POKE) maintenant (PEEK(53272)AND240)+12. Rien ne se passe . . . ou presque rien! Le Commodore 64 reçoit maintenant ses informations de caractères de la mémoire vive RAM au lieu de la mémoire morte ROM. Comme nous avons cependant copié avec exactitude les caractères de la mémoire morte ROM, on ne voit aucune différence . . . pour le moment!

On peut maintenant facilement changer les caractères. Effacer l'écran et taper un signe@. Faire descendre le curseur de deux lignes et taper:

FOR I = 12288 TO 12288+7:POKE I, 255 – PEEK(I) : NEXT

On vient de créer un signe @ en vidéo inverse!

> **CONSEIL:** Les caractères inverses ne sont que des caractères dont la configuration de bits en mémoire de caractères est inversée.

Ramener maintenant le curseur au programme et appuyer de nouveau sur **RETURN** pour réinverser le caractère (le ramener à la normale). À l'aide de la table des codes d'affichage d'écran, on peut déterminer où se trouve chaque caractère en mémoire vive RAM. Il suffit de se rappeler que le stockage de chaque caractère prend huit positions de mémoire. Voici quelques exemples:

CARACTÈRE	CODE D'AFFICHAGE	POSITION DE DÉPART COURANTE EN MÉMOIRE RAM
@	0	12288
A	1	12296
!	33	12552
>	62	12784

Pour le moment, nous n'avons pris que les 64 premiers caractères. Que doit-on faire si l'on veut l'un des autres caractères.

Qu'arrive-t-il si l'on veut le caractère numéro 154, qui est le Z inverse? On peut le créer soi-même en inversant un Z ou on peut copier le jeu de caractères inverses à partir de la mémoire morte ROM; on peut aussi prendre le caractère désiré de la mémoire morte ROM et remplacer l'un des caractères dont on n'a pas besoin dans la mémoire vive RAM.

Supposons que l'on décide ne pas avoir besoin du signe >. Remplaçons ce signe par le Z inverse. Taper:

```
FOR I=0 TO 7: POKE 12784 + I, 255-PEEK(I+12496): NEXT
```

Taper maintenant un signe >. Il est remplacé par un Z inverse. On peut maintenant taper autant de fois le signe > qu'on le désire, on obtient toujours un Z inverse (ce changement n'est en fait qu'une illusion, car le signe > ressemble peut-être à un Z inverse, mais il se conduit toujours comme un > dans un programme. Essayer un programme avec un signe >; il fonctionne toujours normalement, mais son aspect est bizarre.)

Révision rapide: on peut maintenant copier les caractères de la mémoire morte ROM en mémoire vive RAM. On peut même extraire ceux que l'on désire. Il ne reste plus qu'une opération (la meilleure) avec les caractères programmables . . . créer les siens propres.

Comment les caractères sont-ils stockés en mémoire morte ROM? Chaque caractère occupe un groupe de 8 octets. Les configurations de bits des octets commandent directement le caractère. Si l'on dispose 8 octets les uns sur les autres et si l'on écrit chacun d'eux sous forme de huit chiffres binaires, on obtient une matrice de huit par huit qui ressemble aux caractères. Si un bit est à un, on obtient un point à cette position. Si un bit est à zéro, on obtient un espace vide à cette position.

Quand on crée des caractères, on établit le même genre de table en mémoire. Taper NEW, puis le programme suivant:

```
10 FOR I = 12448 TO 12455 : READ A: POKE I,A: NEXT
20 DATA 60, 66, 165, 129, 165, 153, 66, 60
```

Taper ensuite RUN. Le programme remplace la lettre T par un visage souriant. Taper plusieurs T pour faire apparaître le visage. Chacun des nombres de l'instruction DATA de la ligne 20 correspond à une rangée du visage souriant. La matrice du visage a l'aspect suivant:

	7	6	5	4	3	2	1	0	Binaire	Décimal
Rangée 0			*	*	*	*			00111100	60
1		*					*		01000010	66
2	*		*			*		*	10100101	165
3	*							*	10000001	129
4	*		*			*		*	10100101	165
5	*			*	*			*	10011001	153
6		*					*		01000010	66
Rangée 7			*	*	*	*			00111100	60

	7	6	5	4	3	2	1	0
0								
1								
2								
3								
4								
5								
6								
7								

Figure 3-1. Feuille de travail de caractère programmable.

La feuille de travail de caractère programmable (figure 3-1) permet de préparer soi-même des caractères. La feuille représente une matrice de 8 par 8, avec numéros des rangées et des colonnes. (Si l'on considère chaque rangée comme un mot binaire, les chiffres correspondent à la valeur de cette position binaire. Chaque colonne correspond à une puissance de 2. Le bit le plus à gauche est égal à 128, soit 2 à la puissance 7; la colonne suivante équivaut à 64, soit 2 à la puissance 6 et ainsi de suite jusqu'à la colonne la plus à droite (bit 0) qui est égale à 1 ou 2 à la puissance 0).

Noter un X sur la matrice à chaque position où l'on désire avoir un point dans le caractère. Quand le caractère est prêt, on peut créer l'instruction DATA correspondant à ce caractère.

Commençons par la première rangée. À chaque position avec un X, prendre le chiffre en haut de la colonne (chiffre à la puissance de 2 comme on l'a expliqué ci-dessus) et le noter. Quand on a relevé les chiffres de chaque colonne de la première rangée, les additionner. Noter le nombre obtenu, à côté de la rangée. Ce nombre doit être placé dans l'instruction DATA pour tracer cette rangée.

Faire de même avec les autres rangées (1-7). Quand on a terminé, on doit avoir 8 nombres compris entre 0 et 255. Si l'un des nombres n'est pas dans cet intervalle, vérifier l'addition. Les nombres doivent absolument être tous dans cet intervalle. Si l'on a moins de 8 nombres, on a oublié une rangée. Certaines rangées peuvent avoir un total de 0; elles sont toutes aussi importantes que les autres.

Dans l'instruction DATA de la ligne 20, remplacer les nombres par ceux que l'on vient de calculer et exécuter (RUN) le programme. Taper ensuite un T. Chaque fois que l'on tape un T, on obtient le caractère créé.

Si l'on n'aime pas l'aspect du caractère, il suffit de changer les nombres de l'instruction DATA et d'exécuter (RUN) de nouveau le programme jusqu'à ce qu'on en soit satisfait.

C'est aussi simple que cela!

CONSEIL: Pour arriver à de meilleurs résultats avec les caractères, toujours faire des lignes verticales d'au moins 2 points (bits) de large. On évite ainsi le bruit chromatique (distorsion de la couleur) dans les caractères quand on les affiche sur un écran de visualisation.

Voici un exemple de programme avec des caractères programmables normaux:

```
10 REM * EXEMPLE 1 *
20 REM CREATION DE CARACTERES PROGRAMMABLES
31 POKE56334, PEEK(56334)AND254:POKE1,PEEK(1)AND251:REM ARRET KB ET ENTREE/SORTIE
35 FORI=0TO63:REM INTERVALLE DE CARACTERES A COPIER DE LA MEMOIRE MORTE
36 FORJ=0TO7:REM COPIER LES 8 OCTETS PAR CARACTERE
37 POKE12288+I*8+J,PEEK(53248+I*8*J):REM COPIE D'UN OCTET
38 NEXTJ:NEXTI:REM ALLER A L'OCTET OU CARACTERE SUIVANT
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM MISE EN FONCTION ENTREE/SORTIE ET KB
40 POKE53272,(PEEK(53272)AND240)+12:REM FIXER POINTEUR DE CARACTERE A MEM. 12288
60 FORCHAR=60TO63:REM PROGRAMME DES CARACTERES 60 A 63
80 FORBYTE=0TO7:REM FAIRE LES 8 OCTETS D'UN CARACTERE
100 READ NUMBER:REM LIRE 1/8 DE DONNEE CARACTERE
120 POKE12288+(8*CHAR)+BYTE, NUMERO:REM STOCKE LES DONNEES EN MEMOIRE
140 NEXTBYTE:NEXTCHAR:REM PEUT AUSSI ETRE OCTET, CARACTERE SUIVANT
150 PRINTCHR$(147)TAB(255)CHR$(60);
155 PRINTCHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LA LIGNE 150 MET LES CARACTERES NOUVELLEMENT DEFINIS SUR L'ECRAN
170 GETA$:REM ATTENDRE QUE L'UTILISATEUR PRESSE UNE TOUCHE
180 IFA$=“”THENGOTO170:REM SI AUCUNE TOUCHE N'A ETE PRESSEE, ESSAYER DE NOUVEAU!
190 POKE53272,21:REM RETOUR AUX CARACTERES NORMAUX
200 DATA4,6,7,5,7,7,3,3:REM DONNEES POUR CARACTERE 60
210 DATA 32,96,224,160,224,224,192,192:REM DONNEES POUR CARACTERE 61
220 DATA7,7,7,31,31,95,143,127:REM DONNEES POUR CARACTERE 62
230 DATA 224,224,224,248,248,248,240,224:REM DONNEES POUR CARACTERE 63
240 END
```

CARACTÈRES GRAPHIQUES EN MODE MULTICOLORE

Les caractères graphiques normaux à haute définition permettent la commande de points très petits sur l'écran. Chaque point en mémoire de caractères peut avoir 2 valeurs possibles: 1 en fonction et 0 hors fonction. Quand un point est hors fonction, la couleur de l'écran est utilisée dans l'espace réservé à ce point. Si le point est en fonction, il est de la couleur du caractère que l'on a choisi pour cette position d'écran. Si l'on utilise des caractères graphiques normaux à haute définition, tous les points dans chaque caractère de 8 x 8 peuvent avoir une couleur d'arrière-plan ou une couleur de premier plan. On limite ainsi à un certain degré la définition de la couleur dans cet espace. Par exemple, on peut rencontrer des problèmes quand deux lignes de couleur différentes se coupent.

Le mode multicolore permet de résoudre ce problème. En mode multicolore, on dispose d'un choix de 4 couleurs pour chaque point: couleur d'écran (registre 0 pour couleur d'arrière-plan), couleur du registre d'arrière-plan n° 1, couleur du registre de couleur de fond n° 2 ou couleur du caractère. On doit cependant faire une concession à la définition horizontale, car chaque point en mode multicolore est deux fois plus large qu'un point à haute définition. Cette perte minime de définition est largement compensée par les autres possibilités du mode multicolore.

BIT DE MODE MULTICOLORE

Pour passer en mode multicolore, fixer le bit 4 du registre de commande de VIC-II à la position 53270 ($DO16) à 1 à l'aide de l'instruction POKE suivante:

POKE 53270,PEEK(53270)OR 16

Pour arrêter le mode multicolore, mettre le bit 4 de la position 53270 à 0 avec l'instruction POKE suivante:

POKE 53270,PEEK(53270)AND 239

Le mode multicolore est mis en et hors fonction pour chaque espace de l'écran de façon qu'on puisse mélanger des graphiques multicolores avec les graphiques à haute définition. Cette disposition est commandée par le bit 3 de la mémoire de couleurs. Cette mémoire commence à la position 55296 ($D800 en hexadécimal). Si le chiffre en mémoire de couleurs est inférieur à 8 (0-7), l'espace correspondant de l'écran vidéo est en haute définition normale, dans la couleur (0 à 7) choisie. Si le nombre en mémoire de couleurs est supérieur ou égal à 8 (de 8 à 15), l'espace est alors affiché en mode multicolore.

En écrivant (POKE) un nombre dans la mémoire de couleurs, on peut changer la couleur du caractère à cette position sur l'écran. Si l'on écrit (POKE) un chiffre de 0 à 7, on obtient les couleurs de caractères normaux. Si l'on écrit (POKE) un nombre de 8 à 15, on met l'espace en mode multicolore. En mettant le BIT 3 en fonction dans la mémoire de couleur, on obtient donc le mode multicolore. En mettrant le BIT 3 hors fonction, on obtient le mode normal à haute définition.

Quand le mode multicolore est fixé dans un espace, les bits du caractère déterminent les couleurs affichées pour les points. Voici par exemple une image de la lettre A et sa configuration de bits:

IMAGE	CONFIGURATION DE BITS
**	00011000
*****	00111100
** **	01100110
******	01111110
** **	01100110
** **	01100110
** **	01100110
	00000000

En mode normal ou à haute définition, la couleur de l'écran est affichée aux points correspondant à un bit 0; la couleur du caractère est affichée aux points correspondant à un bit 1. Le mode multicolore utilise les bits par paires, de la façon suivante:

IMAGE	CONFIGURATION DE BITS
AABB	00 01 10 00
CCCC	00 11 11 00
AABBAABB	01 10 01 10
AACCCCBB	01 11 11 10
AABBAABB	01 10 01 10
AABBAABB	01 10 01 10
AABBAABB	01 10 01 10
	00 00 00 00

Dans la zone d'image ci-dessus, les espaces marqués AA ont la couleur d'arrière-plan n° 1, les espaces marqués BB la couleur d'arrière-plan n° 2 et les espaces marqués CC la couleur du caractère. Les paires de bits déterminent cette configuration d'après le tableau suivant:

PAIRE DE BITS	REGISTRE DE COULEUR	POSITION
00	Couleur d'arrière-plan n° 0 (couleur de l'écran)	53281 ($D021)
01	Couleur d'arrière-plan n° 1	53282 ($D022)
10	Couleur d'arrière-plan n° 2	53283 ($D023)
11	Couleur spécifiée par les 3 bits inférieurs de la mémoire de couleurs	mémoire vive de couleurs

REMARQUE: La couleur de premier plan d'un caractère graphique programmable est 10. La couleur de premier plan de caractère est 11.

Taper NEW puis le programme de démonstration suivant:

```
100 POKE53281,1:REM FIXE LA COULEUR D'ARRIERE-PLAN # 0 AU BLANC
110 POKE53282,3:REM FIXE LA COULEUR D'ARRIERE-PLAN # 1 AU TURQUOISE
120 POKE53283,8:REM FIXE LA COULEUR D'ARRIERE-PLAN # 2 A L'ORANGE
130 POKE53270,PEEK(53270)OR16:REM MET LE MODE MULTICOLORE EN
FONCTION
140 C=13*4096+8*256:REM FIXE C POUR POINTER A LA MEMOIRE DE COULEURS
150 PRINTCHR$(147)"AAAAAAAAA"
160 FORL=0TO9
170 POKEC+L,8:REM UTILISE LE NOIR DE MODE MULTICOLORE
180 NEXT
```

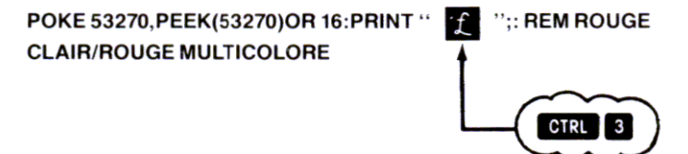
La couleur d'écran est le blanc, la couleur de caractère le noir, une couleur de registre le turquoise (bleu vert) et l'autre l'orange.

On ne met pas réellement des codes de couleur dans l'espace pour la couleur de caractère; on utilise en fait des références aux registres associés à ces couleurs. On conserve ainsi de la mémoire, car on peut utiliser 2 bits pour choisir 16 couleurs (arrière-plan) ou 8 couleurs (caractères). On dispose ainsi de possibilités remarquables. Il suffit de changer l'un des registres indirects pour changer chaque point de cette couleur. De ce fait, on peut changer instantanément tous les points des couleurs de l'écran et d'arrière-plan sur l'écran entier. Voici un exemple de changement du registre n° 1 de couleur d'arrière-plan:

```
100 POKE 53270,PEEK(53270)OR16:REM MISE EN FONCTION DU MODE
MULTICOLORE
110 PRINTCHR$(147)CHR$(18);
                  C= 1
120 PRINT" ▒ ";:REM TAPE C= & 1 POUR ORANGE OU
ARRIERE-PLAN NOIR EN MULTICOLORE
130 FORL=1T022:PRINTCHR$(65);:NEXT
135 FORT=1TO500:NEXT
                  CTRL 7
140 PRINT" ▒ ";:REM TAPER CTRL & 7 POUR CHANGEMENT
DE COULEUR BLEUE
145 FORT=1TO500:NEXT
                  CTRL 1
150 PRINT" ▒ APPUYER SUR UNE TOUCHE"
160 GETA$:IFA$=" "THEN160
170 X=INT(RND(1)*16)
180 POKE 53282,X
190 GOTO 160
```

En utilisant la touche C= et les touches de couleur, on peut faire passer les caractères à n'importe quelle couleur, y compris les caractères multicolores. Par exemple, taper la commande suivante:

POKE 53270,PEEK(53270)OR 16:PRINT " £ ";: REM ROUGE CLAIR/ROUGE MULTICOLORE



Le mot READY (PRÊT) et les autres frappes sont affichés en mode multicolore. Une autre commande de couleur peut ramener au texte normal.

Voici un exemple de programme avec caractères programmables multicolores:

```
10 REM * EXEMPLE 2 *
20 REM CREATION DE CARACTERES PROGRAMMABLES MULTICOLORES
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
35 FORI=0TO63:REM INTERVALLE DE CARACTERES A COPIER DE LA MEMOIRE
MORTE ROM
36 FORJ=0TO7:REM COPIE LES 8 OCTETS PAR CARACTERE
37 POKE12288+I*8+J,PEEK(53248+I*8+J):REM COPIE UN OCTET
38 NEXTJ,I:REM VA A L'OCTET OU CARACTERE SUIVANT
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM MET EN FONCTION
ENTREE/SORTIE ET KB
40 POKE53272,(PEEK53272)AND240)+12:REM FIXE LE POINTEUR DE
CARACTERE A MEM. 12288
50 POKE53270,PEEK(53270)OR16
51 POKE53281,0:REM FIXE LA COULEUR D'ARRIERE-PLAN # 0 AU NOIR
52 POKE53282,2:REM FIXE LA COULEUR D'ARRIERE-PLAN # 1 AU ROUGE
53 POKE53283,7:REM FIXE LA COULEUR D'ARRIERE-PLAN # 2 AU JAUNE
60 FORCHAR=60TO63:REM PROGRAMME CARACTERES 60 A 63
90 FORBYTE=0TO7:REM DONNE LES 8 OCTETS D'UN CARACTERE
100 READNUMBER:REM LIT1/8 DES DONNEES DU CARACTERE
120 POKE12288+(8*CHAR)+BYTE,NUMBER:REM STOCKE LES DONNEES EN
MEMOIRE
140 NEXTBYTE,CHAR
150          SHIFT CLR/HOME
PRINT"  "TAB(255)CHR$(60)CHR$(61)TAB(55)CHR$(62)CHR $(63)
160 REM LA LIGNE 150 MET LES CARACTERES NOUVELLEMENT DEFINIS SUR
L'ECRAN
170 GETA$:REM ATTEND QUE L'UTILISATEUR APPUIE SUR UNE TOUCHE
180 IFA$=""THEN170:REM SI AUCUNE TOUCHE N'A ETE PRESSEE, ESSAYER
DE NOUVEAU
190 POKE53272,21:POKE53270,PEEK(53270)AND239:REM RETOUR AUX
CARACTERES NORMAUX
200 DATA129,37,21,29,93,85,85,85:REM DONNEES POUR LE CARACTERE 60
210 DATA66,72,84,116,117,85,85,85:REM DONNEES POUR LE CARACTERE 61
220 DATA87,87,85,21,8,8,40,0:REM DONNEES POUR LA CARACTERE 62
230 DATA213,213,85,84,32,32,40,0:REM DONNEES POUR LE CARACTERE 63
240 END
```

MODE ÉTENDU DE COULEUR D'ARRIÈRE-PLAN

Le mode étendu de couleur d'arrière-plan permet de commander la couleur d'arrière-plan de chaque caractère individuel, ainsi que !a couleur de premier plan. Ce mode permet ainsi d'afficher un caractère bleu avec arrière-plan jaune sur un écran blanc.

4 registres sont disponibles pour le mode étendu de couleur d'arrière-plan. On dispose d'un choix de 16 couleurs pour chacun des registres.

La mémoire de couleurs sert à maintenir la couleur de premier plan en mode étendu d'arrière-plan. Elle s'utilise comme en mode de caractères normaux.

Le mode étendu de caractères impose cependant une limite au nombre de caractères différents que l'on peut afficher. Quand le mode étendu de couleur est en fonction, on ne peut utiliser que les 64 premiers caractères de la mémoire morte ROM de caractères (ou les 64 premiers caractères du jeu de caractères programmables de l'utilisateur). En effet, deux des bits du code de caractère servent à choisir la couleur d'arrière-plan. Ce mode fonctionne de la façon suivante:

Le code de caractère (nombre que l'on écrit (POKE) sur l'écran) de la lettre "A" est 1. Si l'on écrit (POKE) un 1 sur l'écran quand le mode étendu de couleur est en fonction, un "A" doit apparaître. Si l'on écrit (POKE) 65 sur l'écran, on peut normalement s'attendre à ce que le caractère ayant le code (CHR$) 129 apparaisse; ce caractère est un "A" inverse. Il n'en est rien en mode étendu de couleur; on obtient à la place le même "A" non inversé, mais avec une couleur d'arrière-plan différente. Le tableau suivant donne les codes:

CODE DE CARACTÈRE			REGISTRE DE COULEUR D'ARRIÈRE-PLAN	
INTERVALLE	BIT 7	BIT 6	NOMBRE	ADRESSE
0-63	0	0	0	53281 ($D021)
64-127	0	1	1	53282 ($D022)
128-191	1	0	2	53283 ($D023)
192-255	1	1	3	53284 ($D024)

On met le mode étendu de couleur en fonction en fixant le bit 6 du registre de VIC-II à 1 à la position 53265 ($D011 en hexadécimal). Dans ce but, utiliser l'instruction POKE suivante:

```
POKE 53265,PEEK(53265)OR 64
```

On arrête le mode étendu de couleur en fixant le bit 6 du registre de VIC-II à 0 à la position 53265 ($D011). Dans ce but, utiliser l'instruction POKE suivante:

POKE 53265, PEEK(53265)AND 191

GRAPHIQUES À TOPOGRAPHIE BINAIRE

Si l'on écrit des jeux, si l'on trace des tableaux pour des applications de gestion ou si l'on prépare d'autres types de programmes, on finit tôt ou tard par exiger des affichages à haute définition.

Le Commodore 64 a été conçu dans ce but. On peut obtenir une haute définition sur l'écran par l'intermédiaire de la topographie binaire. En topographie binaire, chaque point (élément d'image) possible de définition sur l'écran reçoit son propre bit (position) en mémoire. Si le bit de mémoire est un, le point attribué est en fonction. Si le bit est zéro, le point est hors fonction.

Les graphiques à haute définition ont quand même quelques inconvénients qui font qu'on ne les utilise pas toujours. Tout d'abord, il faut un grand espace de mémoire pour établir la topographie binaire de l'écran entier. En effet, chaque élément d'image (pixel) doit être commandé par un bit de mémoire. On doit donc avoir un bit de mémoire pour chaque élément d'image (soit 1 octet pour 8 éléments d'image). Chaque caractère ayant 8 par 8 points et l'écran comprenant 40 lignes de 25 caractères chacune, on a une résolution de 320 par 200 éléments d'image (points) pour l'écran complet. On dispose ainsi de 64000 points séparés dont chacun doit avoir un bit en mémoire. Il faut donc 8000 octets de mémoire pour établir la topographie de l'écran entier.

En général, les opérations à haute définition se font en plusieurs programmes de routine répétitifs, courts et simples. Ce genre de processus est malheureusement assez lent si l'on essaie d'écrire des programmes de routine à haute définition en BASIC. Ce sont pourtant exactement ces programmes de routine que le langage machine exécute le mieux. Il faut donc écrire les programmes totalement en langage machine ou appeler des programmes de routine à haute définition en langage machine à partir du programme BASIC en utilisant la commande SYS. On dispose ainsi pour les graphiques de la facilité d'écriture du BASIC et de la rapidité du langage machine. On peut aussi se procurer la cartouche VSP pour ajouter des commandes de haute définition au BASIC du COMMODORE 64.

Aux fins de clarté, tous les exemples de la présente section sont en BASIC. Passons maintenant aux détails techniques.

LA TOPOGRAPHIE BINAIRE est l'une des techniques de caractères graphiques les plus utilisées en informatique. Elle permet de créer des images très détaillées. Fondamentalement, quand le Commodore 64 passe en mode de topographie binaire, il affiche directement une section de mémoire de 8 K sur l'écran de visualisation. En mode de topographie binaire, on peut décider *directement* si un point particulier de l'écran doit être en ou hors fonction.

Le Commodore 64 permet d'utiliser deux types de topographie binaire qui sont:

1) Le mode normal de topographie binaire (haute définition) avec 320 par 200 points.
2) Le mode multicolore de topographie binaire (définition de 160 par 200 points).

Chacun d'eux est très semblable au type de caractère correspondant; le type normal a une définition plus poussée, mais moins de choix de couleurs. Par ailleurs, la topographie binaire multicolore a une définition horizontale moins poussée compensée par un plus grand nombre de couleurs dans un carré de 8 x 8 points.

MODE NORMAL DE TOPOGRAPHIE BINAIRE À HAUTE DÉFINITION

Le mode normal de topographie binaire donne une définition de 320 points horizontaux par 200 points verticaux avec un choix de 2 couleurs dans chaque section de 8 par 8 points. On choisit le mode de topographie binaire (mise en fonction) en fixant le bit 5 du registre de commande de VIC-II à 1 à la position 53265 ($D011 en hexadécimal). Dans ce but, utiliser l'instruction POKE suivante:

```
POKE 53265,PEEK(53265)OR 32
```

On coupe le mode de topographie binaire en fixant le bit 5 du registre de commande de VIC-II à zéro à la position 53265 ($D011), avec l'instruction POKE suivante:

```
POKE53265,PEEK(53265)AND 223
```

Avant de passer aux détails du mode de topographie binaire, nous devons encore résoudre le problème de la localisation de la zone de topographie binaire.

FONCTIONNEMENT

Dans la section sur les caractères programmables, on a vu que l'on pouvait fixer la configuration de bits d'un caractère stocké en mémoire vive RAM pratiquement au gré de l'utilisateur. Si, en même temps, on change le caractère affiché sur l'écran, on peut changer un point simple. Cette notion est à la base de la topographie binaire. L'écran entier est rempli de caractères programmables; on peut faire directement les changements dans la mémoire d'où viennent les configurations des caractères programmables.

Chacune des positions de la mémoire d'écran qui a servi à commander le caractère affiché sert maintenant aux informations de couleur. Par exemple, au lieu d'écrire (POKE) un 1 à la position 1024 pour faire apparaître un "A" dans le coin supérieur gauche de l'écran, cette position commande maintenant les couleurs des bits dans cet espace.

En mode de topographie binaire, les couleurs des carrés ne viennent pas de la mémoire de couleurs comme c'est le cas avec les modes de caractères. Les couleurs viennent de la mémoire d'écran. Les 4 bits supérieurs de la mémoire d'écran prennent la couleur des bits fixés à 1 dans la zone de 8 par 8 commandée par cette position de mémoire d'écran. Les 4 bits inférieurs établissent la couleur de tout bit fixé à 0.

EXEMPLE: Taper le programme suivant:

```
5   BASE=2*4096:POKE53272,PEEK(53272)OR8:REM MET LA TOPOGRAPHIE
BINAIRE A 8192
10  POKE53265,PEEK(53265)OR32:REM ENTRE LE MODE DE TOPOGRAPHIE
BINAIRE
```

Exécuter (RUN) le programme.

Des informations parasites apparaissent sur l'écran! Comme en mode normal d'écran, on doit effacer l'écran à haute définition avant de l'utiliser. L'impression de CLR ne donne malheureusement aucun résultat dans ce cas. On doit effacer la section de mémoire que l'on utilise pour les caractères programmables. Presser les touches RUN/STOP et RESTORE, puis ajouter les lignes suivantes au programme pour effacer l'écran à haute définition:

```
20FORI=BASETOBASE+7999:POKEI,0:NEXT:REM EFFACE LA TOPOGRAPHIE
BINAIRE
30 FORI=1024TO2023:POKEI,3:NEXT:REM FIXE LA COULEUR AU TURQUOISE ET
AU NOIR
```

Exécuter (RUN) de nouveau le programme. L'écran doit s'effacer puis la couleur turquoise ou bleu vert doit le couvrir complètement. Nous voulons maintenant remettre les points en et hors fonction sur l'écran à haute définition.

Pour mettre un point en ou hors fonction, on doit pouvoir localiser le bit correct dans la mémoire de caractères qui est fixé à 1. On doit donc trouver le caractère à changer, la rangée de ce caractère et le bit à changer dans cette rangée. Une formule est nécessaire pour ce calcul.

Nous utilisons X et Y pour les positions horizontale et verticale d'un point. Le point qui correspond à X=0 et Y=0 est dans le coin supérieur gauche de l'affichage. Les points vers la droite ont des valeurs de X plus élevées; les points vers le bas ont des valeurs de Y plus élevées. Pour tirer le meilleur parti de la topographie binaire, disposer l'affichage de la façon suivante:

```
0--------------------------------------X--------------------------------------319
.                                                                              .
.                                                                              .
.                                                                              .
Y                                                                              .
.                                                                              .
.                                                                              .
.                                                                              .
199-----------------------------------------------------------------------------
```

Chaque point possède des coordonnées X et Y. Ce format permet de fixer facilement tout point sur l'écran.

Dans la réalité, on dispose en fait des renseignements suivants:

PREMIÈRE LIGNE (RANGÉE 0)	OCTET 0	OCTET 8	OCTET 16	OCTET 24............	OCTET 312
	OCTET 1	OCTET 9	.	.	OCTET 313
	OCTET 2	OCTET 10	.	.	OCTET 314
	OCTET 3	OCTET 11	.	.	OCTET 315
	OCTET 4	OCTET 12	.	.	OCTET 316
	OCTET 5	OCTET 13	.	.	OCTET 317
	OCTET 6	OCTET 14	.	.	OCTET 318
	OCTET 7	OCTET 15	.	.	OCTET 319
DEUXIÈME LIGNE (RANGÉE 1)	OCTET 320	OCTET 328	OCTET 336	OCTET 344..........	OCTET 632
	OCTET 321	OCTET 329	.	.	OCTET 633
	OCTET 322	OCTET 330	.	.	OCTET 634
	OCTET 323	OCTET 331	.	.	OCTET 635
	OCTET 324	OCTET 332	.	.	OCTET 636
	OCTET 325	OCTET 333	.	.	OCTET 637
	OCTET 326	OCTET 334	.	.	OCTET 638
	OCTET 327	OCTET 335	.	.	OCTET 639

Les caractères programmables qui constituent la topographie binaire sont disposés en 25 rangées de 40 colonnes. Cette méthode est idéale avec un texte, mais elle rend la topographie binaire assez difficile. (Cependant, elle se justifie parfaitement; voir la section "Mélange des modes".)

La formule suivante facilite la commande des points sur l'écran de topographie de mémoire:

Le début de la zone de mémoire d'affichage s'appelle la base. Le numéro de rangée (de 0 à 24) du point est:

ROW = INT(Y/8) (Il y a 320 octets par ligne.)

La position de caractère de cette ligne (de 0 à 39) est:

CHAR = INT(X/8) (Il y a 8 octets par caractère.)

La ligne de cette position de caractère (de 0 à 7) est:

LINE = Y AND 7

Le bit de cet octet est:

BIT = 7 −(X AND 7)

Réunissons maintenant ces formules. L'octet où se trouve le point de mémoire de caractère (X et Y) se calcule ainsi:

BYTE = BASE + ROW*320 + CHAR*8 + LINE

Pour mettre un bit en fonction sur la grille aux coordonnées X et Y, utiliser la ligne suivante:

POKE BYTE,PEEK(BYTE) OR 2↑BIT

Ajoutons ces calculs au programme. Dans l'exemple suivant, le COMMODORE 64 trace une sinusoïdale:

```
50 FORX=0TO319STEP .5:REM L'ONDE REMPLIT L'ECRAN
60 Y=INT(90+80*SIN(X/10))
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LN=YAND7
90 BY=BASE+RO*320+8*CH+LN
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2↑BI)
120 NEXTX
125 POKE1024,16
130 GOTO130
```

À la ligne 60, le calcul change les valeurs de la fonction sinusoïdale d'un intervalle de +1 à −1 à un intervalle de 10 à 170. Les lignes 70 à 100 calculent le caractère, la rangée, l'octet et le bit affectés en utilisant la formule indiquée ci-dessus. La ligne 125 indique que le programme est terminé en changeant la couleur du coin supérieur gauche de l'écran. La ligne 130 fige le programme en le mettant dans une boucle infinie. Quand on ne désire plus observer l'affichage, appuyer sur la touche **RUN/STOP** et la maintenir, puis appuyer sur la touche **RESTORE** .

Dans un autre exemple, on peut modifier le programme d'onde sinusoïdale pour afficher un demi-cercle. Taper les lignes suivantes pour apporter les changements:

```
50 FORX=0TO160:REM COUVRE LA MOITIE DE L'ECRAN
55 Y1=100+SQR(160*X-X*X)
56 Y2=100-SQR(160*X-X*X)
60 FORY=Y1TOY2STEPY1-Y2
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LN=YAND7
90 BY=BASE+RO*320+8*CH+LN
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2↑BI)
114 NEXT
```

On crée ainsi un demi-cercle dans la zone à haute définition de l'écran.

AVERTISSEMENT: Les variables BASIC peuvent recouvrir l'écran à haute définition. Si l'on a besoin d'un plus grand espace de mémoire, déplacer le bas du BASIC au-dessus de la zone d'écran à haute définition. On peut aussi déplacer cette zone d'écran. Ce problème ne se présente pas en langage machine; on ne le rencontre qu'en écrivant des programmes en BASIC.

MODE MULTICOLORE DE TOPOGRAPHIE BINAIRE

Comme les caractères en mode multicolore, le mode multicolore de topographie binaire permet d'afficher jusqu'à quatre couleurs différentes dans chaque section de 8 par 8 de la topographie binaire. Comme en mode de caractère multicolore, on doit aussi accepter des restrictions de définition horizontale (elle passe de 320 à 160 points).

Dans ce mode, on utilise une section de mémoire de 8 K pour la topographie binaire. En mode multicolore de topographie binaire, on choisit les couleurs dans: (1) le registre de couleur d'arrière-plan 0 (couleur d'arrière-plan d'écran), (2) la matrice vidéo (les 4 bits supérieurs donnent une couleur possible et les 4 bits inférieurs une autre) et (3) la mémoire de couleurs.

Le mode multicolore de topographie binaire se met en fonction en fixant le bit 5 de la position 53265 ($D011) et le bit 4 de la position 53270 ($D016) à 1. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53265,PEEK(53265)OR 32:POKE 53270,PEEK(53270)OR 16

Le mode multicolore de topographie binaire se met hors fonction en fixant le bit 5 de la position 53265 ($D011) et le bit 4 de la position 53270 ($D016) à 0. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53265,PEEK(53265)AND 223:POKE 53270,PEEK(53270)AND 239

Comme en mode de topographie binaire normal à haute définition, il existe une correspondance exacte entre la section de mémoire de 8 K utilisée pour l'affichage et les éléments qui apparaissent sur l'écran. Les points horizontaux ont cependant 2 bits de large. Dans la zone de mémoire d'affichage, 2 bits forment un point pour lequel on dispose d'un choix de 4 couleurs.

BITS	ORIGINE DES INFORMATIONS DE COULEUR
00	Registre n° 0 de couleur d'arrière-plan (couleur d'écran)
01	4 bits supérieurs de la mémoire d'écran
10	4 bits inférieurs de la mémoire d'écran
11	Demi-octet couleur (4 bits)

DÉFILEMENT UNIFORME

La microplaquette VIC-II permet le défilement uniforme dans les sens horizontal et vertical. Un défilement uniforme s'obtient par le déplacement d'un élément d'image de l'écran dans un sens. L'élément peut se déplacer vers le haut ou le bas, vers la gauche ou la droite. Il permet de déplacer uniformément les informations nouvelles sur l'écran, tout en sortant en douceur les caractères par l'autre côté.

La microplaquette VIC-II se charge de la majeure partie du travail, mais l'on doit programmer le défilement réel en langage machine. La microplaquette VIC-II permet de placer l'écran vidéo dans un choix de 8 positions horizontales et de 8 positions verticales. Les registres de défilement de la VIC-II commandent le positionnement. La VIC-II est dotée d'un mode de 38 colonnes et d'un mode de 24 rangées. Les dimensions plus petites d'écran servent à assurer une position de départ de défilement des données nouvelles.

Les opérations suivantes permettent d'arriver à un défilement uniforme:

1) Diminuer les dimensions de l'écran (le cadre s'agrandit).
2) Régler le registre de défilement au maximum (ou à la valeur minimale, suivant le sens du défilement).

3) Mettre les nouvelles données dans la partie appropriée (couverte) de l'écran.
4) Augmenter (ou diminuer) le registre de défilement jusqu'à ce qu'il arrive à la valeur maximale (ou minimale).
5) À ce point, utiliser le programme de routine en langage machine pour déplacer l'écran entier d'un caractère complet dans le sens du défilement.
6) Revenir à l'étape 2.

Pour passer au mode de 38 colonnes, fixer le bit 3 de la position 53270 ($D016) à 0. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53270,PEEK(53270)AND 247

Pour revenir au mode à 40 colonnes, fixer le bit 3 de la position 53270 ($D016) à 1. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53270,PEEK(53270)OR 8

Pour passer en mode à 24 rangées, fixer le bit 3 de la position 53265($D011) à 0. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53265,PEEK(53265)AND 247

Pour revenir au mode à 25 rangées, fixer le bit 3 de la position 53265 ($D011) à 1. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53265,PEEK(53265)OR 8

Pour le défilement dans le sens X, il faut mettre la microplaquette VIC-II en mode à 38 colonnes. On dispose ainsi de place pour le défilement des nouvelles données. En défilement vers la gauche, placer les nouvelles données à droite. En défilement vers la droite, placer les nouvelles données à gauche. Il faut remarquer qu'il reste 40 colonnes en mémoire d'écran, mais 38 seulement sont visibles.

Pour le défilement dans le sens Y, il faut placer la microplaquette VIC-II en mode à 24 rangées. En défilement vers le haut, mettre les données nouvelles dans la dernière rangée, En défilement vers le bas, mettre les données nouvelles dans la première rangée. Au contraire du défilement horizontal où des zones de l'écran sont couvertes de chaque côté, il n'y a qu'une seule zone couverte en défilement vertical. Quand le registre de défilement vertical est fixé à 0, la première ligne est couverte, prête à recevoir des données nouvelles. Quand le registre de défilement vertical est fixé à 7, la dernière rangée est couverte.

Pour le défilement dans le sens X, le registre de défilement se trouve aux bits 2 et 0 du registre de commande VIC-II, à la position 53270 ($D016 en hexadécimal). Comme à l'habitude, il importe de n'affecter que ces deux bits. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53270,(PEEK(53270)AND 248) + X

dans laquelle X correspond à la position horizontale de 0 à 7 sur l'écran.

Pour le défilement dans le sens Y, le registre de défilement se trouve aux bits 2 et 0 du registre de commande VIC-II, à la position 53265 ($D011 en hexadécimal). Comme à l'habitude, il importe de n'affecter que ces deux bits. Dans ce but, utiliser l'instruction POKE suivante.

POKE53265, (PEEK(53265)AND248) + Y

dans laquelle Y correspond à la position verticale de 0 à 7 sur l'écran.

Pour faire défiler le texte sur l'écran à partir du bas, on fait progresser les 3 bits inférieurs de la position 53265 de 0 à 7, on introduit davantage de données sur la ligne couverte au bas de l'écran et on répète ensuite le processus. Pour faire défiler les caractères sur l'écran de la gauche vers la droite, on fait progresser les 3 bits inférieurs de la position 53270 de 0 à 7, on imprime ou écrit (POKE) une autre colonne de données nouvelles dans la colonne 0 de l'écran, puis on répète le processus.

Si l'on fait progresser les bits de défilement de – 1, le texte se déplace dans l'autre sens.

EXEMPLE: Défilement du texte vers le bas de l'écran:

```
10 POKE53265,PEEK(53265)AND 247                    :REM PASSE
EN MODE A 24 RANGEES
20 PRINTCHR$(147)                                  :REM
EFFACE L'ECRAN
30 FORX = 1TO24:PRINTCHR$(17);:NEXT                :REM DEPLACE
LE CURSEUR VERS LE BAS
40 POKE53265,(PEEK(53265)AND248) + 7:PRINT         :REM
POSITION POUR LE PREMIER DEFILEMENT
50 PRINT"        BONJOUR";
60 FORP = 6TO0STEP - 1
70 POKE53265,(PEEK(53265)AND248) + P
80 FORX = 1TO50:NEXT                               :REM
BOUCLE DE RETARD
90 NEXT:GOTO40
```

CARACTÈRES GRAPHIQUES PROGRAMMABLES

Un caractère graphique programmable est un type spécial de caractère définissable par l'utilisateur que l'on peut afficher en tout point de l'écran. Les caractères graphiques programmables sont directement mis à jour par la microplaquette VIC-II. Avec ces caractères, il suffit d'indiquer leur aspect, leur couleur et leur position; la microplaquette VIC-II se charge du reste! Il existe un choix de 16 couleurs pour les caractères graphiques programmables.

On peut utiliser ces caractères avec tous les autres modes graphiques: topographie binaire, caractères, multicolore, etc. Ils conservent toujours leur forme. Un caractère graphique programmable possède la définition de couleur, le mode (haute définition ou multicolore) et la forme qui lui sont propres.

La microplaquette VIC-II peut supporter automatiquement jusqu'à 8 caractères graphiques programmables à la fois. On peut afficher davantage de caractères graphiques programmables avec les techniques d'interruption de trame.

Les caractères graphiques programmables présentent les caractéristiques suivantes:

1) Taille de 24 points horizontaux par 21 points verticaux.
2) Commande individuelle de couleur pour chaque caractère graphique programmable.
3) Mode multicolore de caractère graphique programmable.
4) Grossissement de 2 dans les sens horizontal et(ou) vertical.
5) Choix de la priorité du caractère graphique programmable sur l'arrière-plan.
6) Priorités fixes d'un caractère graphique programmable à un autre.
7) Détection de collision entre caractères graphiques programmables.
8) Détection de collision entre caractère graphique programmable et arrière-plan.

Ces caractéristiques spéciales des caractères graphiques programmables simplifient la programmation de jeux d'aspect très réaliste. Les caractères graphiques programmables étant soutenus par le matériel, il est même possible d'écrire un jeu de qualité en BASIC.

8 caractères graphiques programmables, numérotés de 0 à 7, sont soutenus directement par la microplaquette VIC-II. Chaque caractère graphique programmable possède la position de définition, les registres de position et le registre de couleur qui lui sont propres; il possède aussi ses propres bits de validation et de détection de collision.

NUMÉRO DE COLONNE	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
BIT	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
VALEURS DES DONNÉES DE BITS (MARCHE = 1xVALEUR)	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
RANGÉE 0																								
RANGÉE 1																								
RANGÉE 2																								
RANGÉE 3																								
RANGÉE 4																								
RANGÉE 5																								
RANGÉE 6																								
RANGÉE 7																								
RANGÉE 8																								
RANGÉE 9																								
RANGÉE 10																								
RANGÉE 11																								
RANGÉE 12																								
RANGÉE 13																								
RANGÉE 14																								
RANGÉE 15																								
RANGÉE 16																								
RANGÉE 17																								
RANGÉE 18																								
RANGÉE 19																								
RANGÉE 20																								

Figure 3-2. Bloc de définition de caractère graphique programmable.

DÉFINITION D'UN CARACTÈRE GRAPHIQUE PROGRAMMABLE

Ces caractères graphiques programmables étant de taille plus grande, il faut davantage d'octets pour leur définition. Un caractère graphique programmable occupe 24 par 21 points, soit 504 points. Il faut donc 63 octets (504/8 bits) pour définir un caractère graphique programmable. Ces 63 octets sont disposés en 21 rangées de 3 octets chacune. Une définition de caractère graphique programmable se présente ainsi:

OCTET 0	OCTET 1	OCTET 2
OCTET 3	OCTET 4	OCTET 5
OCTET 6	OCTET 7	OCTET 8
..	..	..
..	..	..
..	..	..
OCTET 60	OCTET 61	OCTET 62

Pour se faire une idée de la création d'un caractère graphique programmable, on peut jeter un coup d'oeil au bloc de définition qui descend au niveau du bit. Voir la figure 3-2.

Dans un caractère graphique programmable normal (à haute définition), chaque bit fixé à 1 est affiché dans la couleur de premier plan du caractère graphique programmable concerné. Chaque bit fixé à 0 est transparent et affiche les données situées derrière lui. Cette condition est analogue à celle d'un caractère normal.

Les caractères graphiques programmables multicolores sont analogues aux caractères multicolores. La baisse de définition horizontale est compensée par une meilleure définition de couleur. La définition du caractère graphique programmable est de 12 points horizontaux par 21 points verticaux. Chaque point du caractère graphique programmable est deux fois plus large, mais le nombre de couleurs est porté à 4 avec ce genre de caractère.

POINTEURS DE CARACTÈRE GRAPHIQUE PROGRAMMABLE

La définition de chaque caractère graphique programmable ne prend que 63 octets, mais il faut un octet de plus à la fin de chacun d'eux comme maintien de place. Chaque caractère graphique programmable prend donc 64 octets. Ce nombre facilite les calculs, car 64 octets correspondent à un nombre pair et à une puissance paire en numération binaire.

Chacun des 8 caractères programmables possède un octet associé ou pointeur de caractère graphique programmable. Les pointeurs déterminent la position de la définition de chaque caractère graphique programmable en mémoire. Ces 8 octets sont tou-

jours à la position des 8 derniers octets du groupe de 1 K de la mémoire d'écran. Avec le Commodore 64, ces octets commencent normalement à la position 2040 ($07F8 en hexadécimal). Si l'on déplace cependant l'écran, la position des pointeurs de caractère graphique programmable se déplace également.

Chaque pointeur de caractère graphique programmable peut contenir un nombre de 0 à 255. Ce numéro est pointé sur la définition du caractère graphique programmable correspondant. Chaque définition de caractère graphique occupant 64 octets, le pointeur peut "voir" en tout point du bloc de mémoire de 16 K, accessible par la microplaquette VIC-II (256*64=16 K).

Si le pointeur n° 0 à la position 2040 contient par exemple le numéro 14, le caractère graphique programmable 0 sera affiché en utilisant les 64 octets, en commençant à la position 14*64 = 896 qui correspond au tampon de cassette. Ce raisonnement est plus clair avec la formule suivante:

POSITION = (BLOC * 16384) + (VALEUR DE POINTEUR DE CARACTÈRE GRAPHIQUE PROGRAMMABLE * 64)

dans laquelle le bloc correspond à la section de mémoire de 16 K que la microplaquette VIC-II observe et qui va de 0 à 3.

La formule ci-dessus donne le début des 64 octets du bloc de définition du caractère graphique programmable.

Si la VIC-II observe le bloc 0 ou le bloc 2, une image de mémoire morte ROM du jeu de caractères est présente à certaines positions, comme nous l'avons précédemment indiqué. On ne peut pas placer de définition de caractère programmable à ces endroits. Si, pour une raison quelconque, on a besoin de plus de 128 définitions différentes de caractères graphiques programmables, on doit utiliser un des blocs sans l'image 1 ou 3 de mémoire morte ROM.

MISE EN FONCTION DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

À la position 53269 ($D015 en hexadécimal) se trouve le registre de validation de caractère graphique programmable de la VIC-II. Chaque caractère graphique programmable possède un bit dans ce registre qui commande sa mise en ou hors fonction. Le registre a l'aspect suivant:

$D015   7 6 5 4 3 2 1 0

Par exemple, pour mettre le caractère graphique programmable 1 en fonction, on doit mettre ce bit à 1. Dans ce but, utiliser l'instruction POKE suivante:

POKE 53269,PEEK(53269)OR 2

L'instruction suivante est plus générale:

POKE 53269,PEEK(53269)OR(2↑SN)

dans laquelle SN correspond au numéro de caractère graphique programmable, de 0 à 7.

REMARQUE: Un caractère graphique programmable doit être mis en fonction avant qu'on puisse le voir.

MISE HORS FONCTION DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

On met un caractère graphique programmable hors fonction en fixant à 0 son bit dans le registre de commande de VIC-II, à la position 53269 ($D015 en hexadécimal). Dans ce but, utiliser l'instruction POKE suivante:

POKE 53269,PEEK(53269)AND(255−2↑SN)

dans laquelle SN correspond au numéro de caractère graphique programmable, de 0 à 7.

COULEURS

Un caractère graphique programmable peut recevoir l'une des 16 couleurs créées par la microplaquette VIC-II. Chaque caractère graphique programmable possède son propre registre de couleur. Nous donnons ci-dessous les positions de mémoire des registres de couleur:

ADRESSE		DESCRIPTION
53287	($D027)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 0
53288	($D028)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 1
53289	($D029)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 2
53290	($D02A)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 3
53291	($D02B)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 4
53292	($D02C)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 5
53293	($D02D)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 6
53294	($D02E)	REGISTRE DE COULEUR, CARACTÈRE GRAPHIQUE PROGRAMMABLE 7

Tous les points du caractère graphique programmable sont affichés dans la couleur contenue dans le registre de couleur du caractère. Le reste du caractère graphique programmable est transparent et laisse voir les éléments qui se trouvent derrière.

MODE MULTICOLORE

Le mode multicolore permet de disposer de 4 couleurs différentes dans chaque caractère graphique programmable. Toutefois, la définition horizontale est réduite de moitié, comme avec les autres modes multicolores. Quand on travaille avec les caractères graphiques en mode multicolore (comme en mode multicolore de caractères), on dispose donc de 12 paires de points au lieu de 24 sur la largeur du caractère. Chaque paire de points est dite paire binaire. Pour le choix des couleurs des points, il faut imaginer chaque paire binaire (paire de points) comme point unique dans le caractère graphique global. La table ci-dessous donne les valeurs des paires binaires nécessaires pour mettre en fonction chacune des quatre couleurs choisies pour le caractère graphique programmable:$

PAIRE BINAIRE	DESCRIPTION
00	TRANSPARENT, COULEUR DE L'ÉCRAN
01	REGISTRE N° 0 MULTICOLORE DE CARACTÈRE GRAPHIQUE PROGRAMMABLE (53285)($D025)
10	REGISTRE DE COULEUR DE CARACTÈRE GRAPHIQUE PROGRAMMABLE
11	REGISTRE N° 1 MULTICOLORE DE CARACTÈRE GRAPHIQUE PROGRAMMABLE (53286)($D026)

REMARQUE: La couleur de premier plan de caractère graphique programmable correspond à 10. Le premier plan de caractère correspond à 11.

CARACTÈRE GRAPHIQUE PROGRAMMABLE EN MODE MULTICOLORE

Pour faire passer un caractère graphique programmable en mode multicolore, mettre en fonction le registre de commande de VIC-II à la position 53276 ($D01C). Dans ce but, utiliser l'instruction POKE suivante:

POKE 53276,PEEK(53276) OR (2↑SN)

dans laquelle SN correspond au numéro du caractère graphique programmable, de 0 à 7.

Pour faire sortir un caractère graphique programmable du mode multicolore, mettre hors fonction le registre de commande VIC-II à la position 53276 ($D01C). Dans ce but, utiliser l'instruction POKE suivante:

POKE 53276,PEEK(53276) AND (255–2↑SN)

dans laquelle SN correspond au numéro du caractère graphique programmable, de 0 à 7.

CARACTÈRES GRAPHIQUES PROGRAMMABLES ÉTENDUS

La VIC-II permet d'étendre un caractère graphique programmable dans le sens vertical et(ou) horizontal. Dans un caractère graphique programmable étendu, chaque point est deux fois plus large ou deux fois plus haut. La définition ne devient pas réellement meilleure; seules les dimensions du caractère graphique programmable augmentent.

Pour étaler un caractère graphique programmable dans le sens horizontal, mettre en fonction (à 1) le bit correspondant dans le registre de commande du VIC-II, à la position 53277 ($D01D en hexadécimal). L'instruction POKE suivante étale un caractère graphique programmable dans le sens horizontal:

POKE 53277,PEEK(53277)OR (2↑SN)

dans laquelle SN correspond au numéro de caractère graphique programmable, de 0 à 7.

Pour ramener un caractère graphique programmable à la taille normale dans le sens horizontal, mettre à 0 le bit correspondant dans le registre de commande de VIC-II, à la position 53277 ($D01D en hexadécimal). L'instruction POKE suivante ramène un caractère graphique programmable à la normale dans le sens horizontal:

POKE 53277,PEEK(53277)AND(255–2↑SN)

dans laquelle SN correspond au numéro de caractère graphique programmable, de 0 à 7.

Pour étendre un caractère graphique programmable dans le sens vertical, mettre à 1 le bit correspondant du registre de commande de VIC-II, à la position 53271 ($D017 en hexadécimal). L'instruction POKE suivante étale un caractère graphique programmable dans le sens vertical:

POKE 53271,PEEK(53271)OR(2↑SN)

dans laquelle SN correspond au numéro du caractère graphique programmable, de 0 à 7.

Pour ramener un caractère graphique programmable à la normale dans le sens vertical, mettre à 0 le bit correspondant dans le registre de commande de VIC-II, à la position 53271 ($D017 en hexadécimal). L'instruction POKE suivante ramène un caractère graphique programmable à la normale dans le sens vertical:

POKE 53271,PEEK(53271)AND(255–2↑SN)

dans laquelle SN correspond au numéro du caractère graphique programmable, de 0 à 7.

POSITIONNEMENT DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

Après avoir créé un caractère graphique programmable, on désire certainement le faire se déplacer sur l'écran. Dans ce but, le Commodore 64 possède trois registres de positionnement:

1) REGISTRE DE POSITIONNEMENT X DE CARACTÈRE GRAPHIQUE PROGRAMMABLE
2) REGISTRE DE POSITIONNEMENT Y DE CARACTÈRE GRAPHIQUE PROGRAMMABLE
3) REGISTRE DE POSITIONNEMENT X DU BIT LE PLUS SIGNIFICATIF

Chaque caractère graphique programmable possède un registre de position X, un registre de position Y et un bit dans le registre de position X de bit le plus significatif. Ces registres permettent de positionner les caractères graphiques programmables avec beaucoup de précision. On peut placer un caractère graphique programmable sur 512 positions X possibles et sur 256 positions Y possibles.

Les registres de position X et Y marchent par paires. Les positions des registres X et Y apparaissent dans la topographie de mémoire de la façon suivante: le registre X du caractère programmable 0 est suivi du registre Y de ce même caractère. Viennent ensuite le registre X du caractère graphique programmable 1, le registre Y de ce même caractère, etc. Après les 16 registres X et Y vient le bit le plus significatif de la position X, placé dans son propre registre.

Le tableau suivant donne les positions de chaque registre de position de caractère graphique programmable. Utiliser ces positions à leur emplacement correspondant dans les instructions POKE:

POSITION		
DÉCIMAL	HEXADÉCIMAL	DESCRIPTION
53248	($D000)	REGISTRE DE POSITION X DU CARACTÈRE 0
53249	($D001)	REGISTRE DE POSITION Y DU CARACTÈRE 0
53250	($D002)	REGISTRE DE POSITION X DU CARACTÈRE 1
53251	($D003)	REGISTRE DE POSITION Y DU CARACTÈRE 1
53252	($D004)	REGISTRE DE POSITION X DU CARACTÈRE 2
53253	($D005)	REGISTRE DE POSITION Y DU CARACTÈRE 2
53254	($D006)	REGISTRE DE POSITION X DU CARACTÈRE 3
53255	($D007)	REGISTRE DE POSITION Y DU CARACTÈRE 3
53256	($D008)	REGISTRE DE POSITION X DU CARACTÈRE 4
53257	($D009)	REGISTRE DE POSITION Y DU CARACTÈRE 4
53258	($D00A)	REGISTRE DE POSITION X DU CARACTÈRE 5
53259	($D00B)	REGISTRE DE POSITION Y DU CARACTÈRE 5
53260	($D00C)	REGISTRE DE POSITION X DU CARACTÈRE 6
53261	($D00D)	REGISTRE DE POSITION Y DU CARACTÈRE 6
53262	($D00E)	REGISTRE DE POSITION X DU CARACTÈRE 7
53263	($D00F)	REGISTRE DE POSITION Y DU CARACTÈRE 7
53264	($D010)	REGISTRE DE POSITION X DU BIT LE PLUS SIGNIFICATIF

La position d'un caractère graphique programmable se calcule du coin supérieur gauche de la zone de 24 par 21 points où l'on peut intégrer le caractère graphique programmable. Le nombre de points entrant dans la composition d'un caractère graphique programmable n'a aucune importance. Même avec un caractère graphique programmable de un point que l'on veut placer au centre de l'écran, on doit encore calculer le positionnement exact en commençant au coin supérieur gauche.

POSITIONNEMENT VERTICAL

La détermination des positions dans le sens horizontal est un peu plus difficile que le positionnement vertical; nous allons donc commencer par ce dernier.

On peut programmer séparément 200 positions de point différentes dans le sens Y sur l'écran de visualisation. Les registres de position Y de caractère graphique programmable peuvent recevoir des nombres jusqu'à 255. On dispose donc d'un nombre suffisant de positions de registre pour assurer le déplacement d'un caractère programmable vers le haut et le bas. Il est aussi essentiel de faire apparaître ou disparaître en douceur un caractère programmable de l'écran. Dans ce but, on a besoin de plus de 200 valeurs.

La première valeur commence en haut de l'écran; elle est de 30 dans le sens Y pour un caractère graphique programmable non étendu. Elle est de 9 dans le sens Y pour un graphique étendu. (Chaque point étant deux fois plus haut, cette valeur est justifiée, car la position initiale se calcule toujours du coin supérieur gauche du caractère graphique programmable.)

La première valeur Y qui place un caractère graphique programmable (étendu ou non) entièrement sur l'écran (avec affichage des 21 lignes possibles) est 50.

La dernière valeur Y qui place un caractère graphique programmable non étendu entièrement sur l'écran est 229. La dernière valeur Y qui place un caractère graphique programmable étendu totalement sur l'écran est 208.

La première valeur Y qui place un caractère graphique programmable totalement hors de l'écran est 250.

EXEMPLE:

```
           SHIFT CLR/HOME
1Ø PRINT"♥"                          :REM EFFACE L'ECRAN
2Ø POKE2Ø4Ø,13                       :REM SORT
LES DONNEES DU CARACTERE Ø DU BLOC 13
3Ø FORI=ØTO62:POKE832+I,129:NEXT     :REM ECRIT LES DONNEES DE
CARACTERES PROGRAMMABLES DANS LE BLOC 13 (13*64=832)
4Ø V=53248                           :REM FIXE LE DEBUT DE LA
PUCE VIDEO
5Ø POKEV+21,1                        :REM VALIDE LE CARACTERE
PROGRAMMABLE 1
6Ø POKEV+39,1                        :REM FIXE LA COULEUR DU
CARACTERE Ø
7Ø POKEV+1,1ØØ                       :REM FIXE LA POSITION Y DU
CARACTERE Ø
8Ø POKEV+16,Ø:POKEV,1ØØ              :REM FIXE LA POSITION X DU
CARACTERE Ø
```

POSITIONNEMENT HORIZONTAL

Le positionnement dans le sens horizontal est plus complexe, car on dispose de plus de 256 positions. Un bit supplémentaire ou 9e bit sert à commander la position X. En ajoutant le bit supplémentaire au point nécessaire, un caractère graphique programmable a maintenant 512 positions possibles dans le sens horizontal X (gauche/droite). On dispose ainsi de davantage de combinaisons possibles qu'il n'est possible de voir sur l'écran. Chaque caractère graphique programmable peut avoir une position de 0 à 511. Toutefois, seules les valeurs comprises entre 24 et 343 sont

visibles sur l'écran. Si la position X d'un caractère graphique programmable est supérieure à 255 (à la droite de l'écran), le bit dans le registre de position X de bit le plus significatif doit être fixé à 1 (en fonction). Si la position X d'un caractère graphique programmable est inférieure à 256 (à gauche de l'écran), le registre de position X de bit le plus significatif de ce caractère doit être à 0 (hors fonction). Les bits 0 à 7 du registre de position X du bit le plus significatif correspondent respectivement aux caractères graphiques programmables 0 à 7.

Le programme suivant déplace un caractère graphique programmable sur l'écran:

EXEMPLE:

SHIFT CLR/HOME

```
10 PRINT" "
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV+1,100
80 FORJ=0TO347
90 HX=INT(J/256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX:NEXT
```

Quand on déplace des caractères graphiques programmables étendus sur la gauche de l'écran, dans le sens horizontal X, on doit faire partir le caractère en dehors de l'écran, du côté droit. En effet, un caractère graphique programmable étendu est plus grand que la place de mémoire disponible du côté gauche de l'écran.

EXEMPLE:

SHIFT CLR/HOME

```
10 PRINT " "
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1:POKEV+23,1:POKEV+29,1
70 POKEV+1,100
80 J=488
90 HX=INT(J/256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX
110 J=J+1:IFJ>511THENJ=0
120 IFJ>488ORJ<348GOTO90
```

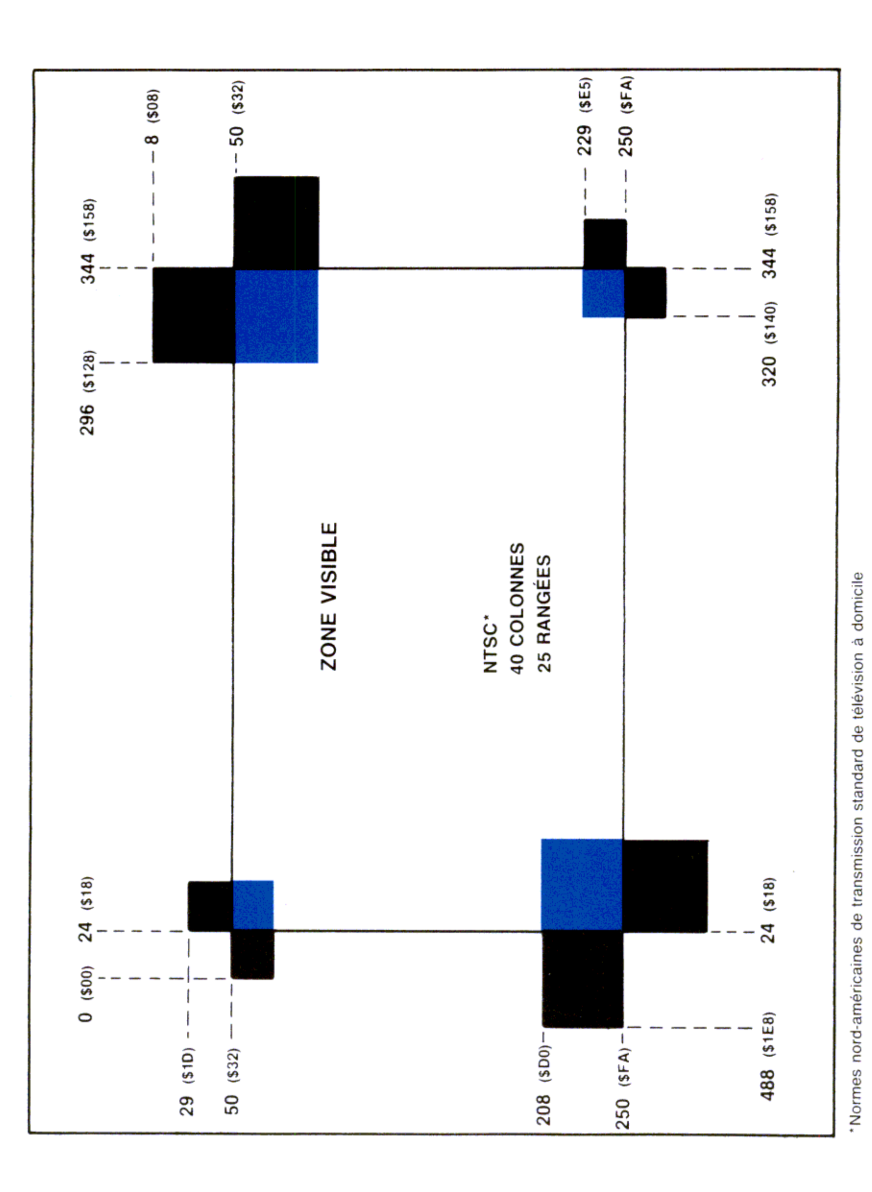
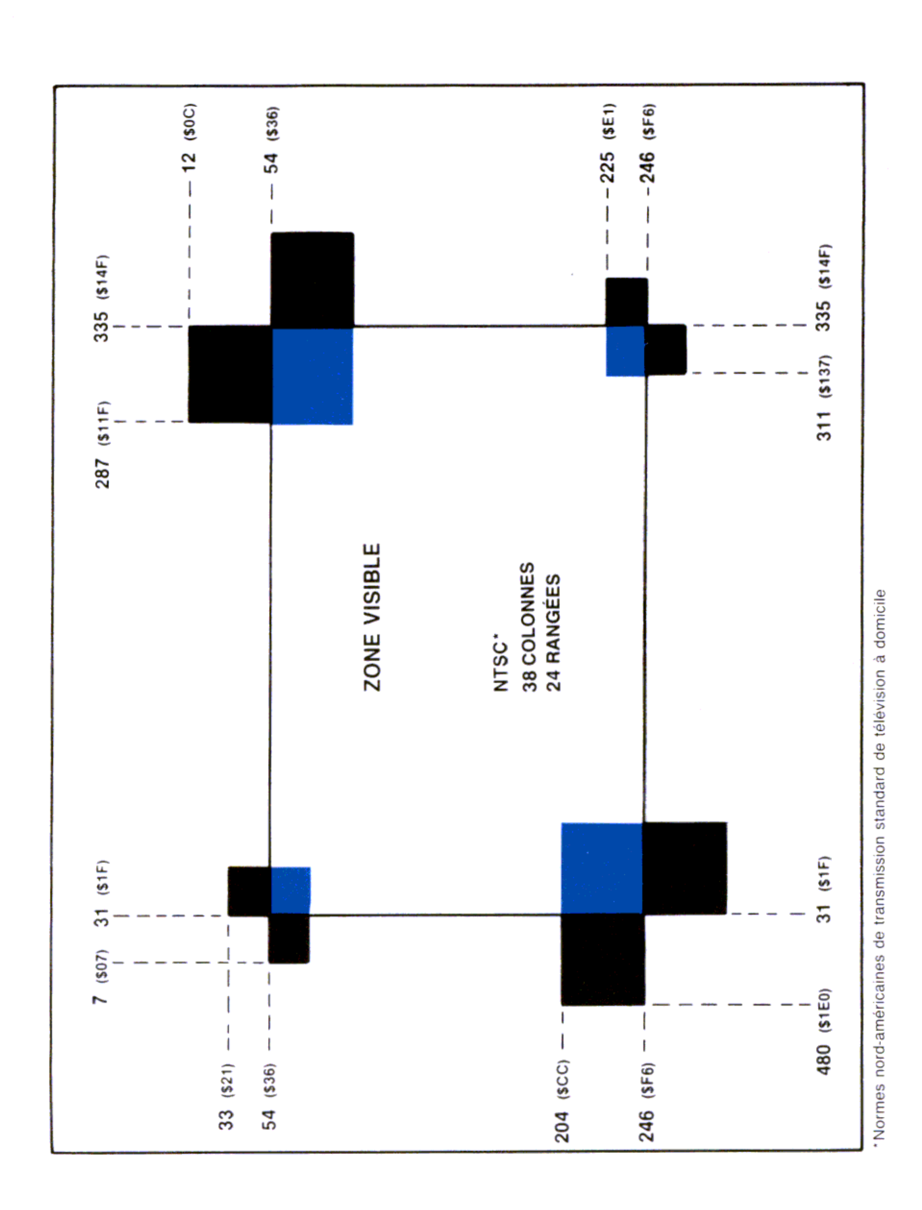


Figure 3-3. Tableaux de positionnement



de caractère graphique programmable

Les tableaux de la figure 3-3 expliquent le positionnement des caractères graphiques programmables.

Ces valeurs permettent de positionner chaque caractère graphique programmable où on le désire. En déplaçant le caractère graphique d'une seule position de point à la fois, on arrive facilement à un déplacement très uniforme.

RÉSUMÉ DU POSITIONNEMENT DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

Les caractères graphiques programmables non étendus sont visibles, au moins partiellement, en mode de 40 colonnes par 25 rangées, dans les limites des paramètres suivants:

$$1 < = X < = 343$$

$$30 < = Y < = 249$$

En mode à 38 colonnes, les paramètres X deviennent:

$$8 < = X < = 334$$

En mode à 24 rangées, les paramètres Y deviennent:

$$34 < = Y < = 245$$

Les caractères graphiques programmables étendus sont visibles, au moins partiellement, en mode de 40 colonnes par 25 rangées dans la limite des paramètres suivants:

$$489 > = X < = 343$$

$$9 > = Y < = 249$$

En mode à 38 colonnes, les paramètres X deviennent:

$$496 > = X < = 334$$

En mode à 24 rangées, les paramètres Y deviennent:

$$13 < = Y < = 245$$

PRIORITÉS D'AFFICHAGE DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

Les caractères graphiques programmables peuvent s'entrecroiser les uns les autres et passer devant ou derrière d'autres objets sur l'écran. Cette caractéristique permet d'obtenir un effet tridimensionnel pour les jeux.

La priorité entre caractères graphiques programmables est fixe. Le caractère graphique programmable 0 a ainsi la priorité la plus élevée; vient ensuite le caractère graphique programmable 1 et ainsi de suite, jusqu'au caractère graphique 7 qui a la priorité la plus basse. Si les caractères graphiques programmables 1 et 6 sont placés de façon qu'ils se croisent, le caractère 1 apparaît donc devant le caractère 6.

Quand on planifie les caractères graphiques programmables qui doivent apparaître au premier plan de l'image, il faut leur donner des numéros plus bas que ceux placés vers l'arrière-plan; ceux-ci doivent recevoir des numéros plus élevés.

REMARQUE: Il est possible d'obtenir un effet de ''fenêtre''. Si un caractère graphique programmable à priorité plus élevée comprend des ''trous'' (zone où les points ne sont pas à 1 et donc hors fonction), on peut voir, par transparence, le caractère graphique programmable de priorité plus basse. On obtient le même résultat avec un caractère graphique programmable et les données d'arrière-plan.

La priorité entre caractères graphiques programmables et arrière-plan se commande à l'aide du registre de priorité caractère graphique programmable/arrière-plan, à la position 53275 ($D01B). Chaque caractère graphique programmable possède un bit dans ce registre. Si ce bit est à 0, le caractère a une priorité plus élevée sur l'écran que celle de l'arrière-plan. Le caractère graphique apparaît donc devant les données d'arrière-plan. Si ce bit est à 1, le caractère graphique programmable a une priorité plus basse que l'arrière-plan; il apparaît alors derrière les données d'arrière-plan.

DÉTECTION DE COLLISION

Les capacités de détection de collision de la microplaquette VIC-II sont particulièrement intéressantes. On peut détecter des collisions entre des caractères graphiques programmables ou entre ces caractères et les données d'arrière-plan. Il se produit une collision quand une partie non nulle d'un caractère graphique recouvre une partie non nulle d'un autre caractère graphique programmable ou autre sur l'écran.

COLLISIONS ENTRE CARACTÈRES GRAPHIQUES PROGRAMMABLES

L'ordinateur signale les collisions entre caractères graphiques programmables dans le registre de collision de caractères graphiques programmables, à la position 53278 ($D01E en hexadécimal), dans le registre de commande de la VIC-II. Chaque caractère graphique programmable possède un bit dans ce registre. Si ce bit est à 1, le caractère graphique programmable correspondant est alors entré en collision. Les bits de ce registre restent fixés jusqu'à la lecture (PEEK). Après la lecture, le registre est automatiquement effacé; il est donc bon de sauvegarder la valeur dans une variable tant qu'on n'a pas fini de l'utiliser.

> **REMARQUE:** Il peut se produire des collisions, même si les caractères graphiques programmables sont en dehors de l'écran.

COLLISIONS ENTRE CARACTÈRES GRAPHIQUES PROGRAMMABLES ET DONNÉES

Les collisions entre caractères graphiques programmables et données sont détectées dans le registre de collision caractères graphiques programmables/données à la position 53279 ($D01F en hexadécimal) du registre de commande de la VIC-II.

Chaque caractère graphique programmable possède un bit dans ce registre. Si ce bit est à 1, le caractère correspondant est alors entré en collision. Les bits de ce registre restent fixés jusqu'à la lecture (PEEK). Après la lecture, le registre est automatiquement effacé; il est donc bon de sauvegarder la valeur dans une variable jusqu'à ce qu'on ait fini de l'utiliser.

> **REMARQUE:** La donnée 01 en mode multicolore est transparente pour les collisions, même si elle apparaît sur l'écran. Quand on établit un écran d'arrière-plan, il est bon de veiller à ne pas provoquer de collision 01 en mode multicolore.

```
10 REM EXEMPLE 1 DE CARACTERES GRAPHIQUES PROGRAMMABLES . . .
20 REM LE BALLON A AIR CHAUD
30 VIC=13*4096:REM LES REGISTRES VIC COMMENCENT ICI
35 POKEVIC+21,1:REM VALIDATION DU CARACTERE GRAPHIQUE PROGRAMMABLE 0
36 POKEVIC+33,14:REM COULEUR D'ARRIERE-PLAN FIXEE A BLEU CLAIR
37 POKEVIC+23,1:REM ETALEMENT Y DU CARACTERE 0
38 POKEVIC+29,1:REM ETALEMENT X DU CARACTERE 0
40 POKE2040,192:REM FIXE LE POINTEUR DU CARACTERE 0
180 POKEVIC+0,100:REM FIXE LA POSITION X DU CARACTERE 0
190 POKEVIC+1,100:REM FIXE LA POSITION Y DU CARACTERE 0
220 POKEVIC+39,1:REM FIXE LA COULEUR DU CARACTERE 0
250 FORY=0TO63:REM COMPTEUR D'OCTET AVEC BOUCLE DE CARACTERE GRAPHIQUE
300 READA:REM LECTURE DANS UN OCTET
310 POKE192*64+Y,A:REM STOCKE LES DONNEES DANS LA ZONE DE CARACTERES GRAPHIQUES
320 NEXTY:REM FERMETURE DE LA BOUCLE
330 DX=1:DY=1
340 X=PEEK(VIC):REM LECTURE DE LA POSITION X DU CARACTERE 0
350 Y=PEEK(VIC+1):REM LECTURE DE LA POSITION Y DU CARACTERE 0
360 IFY=50ORY=208THENDY=-DY:REM SI Y EST AU BORD DE . . .
370 REM L'ECRAN, ALORS DELTA Y INVERSE
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SI LE CARACTERE GRAPHIQUE PROGRAMMABLE . . .
390 REM TOUCHE LE BORD GAUCHE (X=24 ET LE BIT LE PLUS SIGNIFICATIF CARACTERE 0 EST 0), L'INVERSER
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SI LE CARACTERE . . .
410 REM TOUCHE LE BORD DROIT (X=40 ET LE BIT LE PLUS SIGNIFICATIF DU CARACTERE 0 EST 1), L'INVERSER
420 IFX=255ANDDX=1THENX=-1:SIDE=1
430 REM PASSER A L'AUTRE COTE DE L'ECRAN
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM PASSER A L'AUTRE COTE DE L'ECRAN
460 X=X+DX:REM AJOUTER DELTA X A X
470 X=XAND255:REM S'ASSURER QUE X EST DANS L'INTERVALLE PERMIS
480 Y=Y+DY:REM AJOUTER DELTA Y A Y
485 POKEVIC+16,SIDE
```

```
490 POKEVIC,X:REM METTRE LA NOUVELLE VALEUR X DANS LA POSITION X DU
CARACTERE 0
510 POKEVIC + 1,Y:REM METTRE LA NOUVELLE VALEUR Y DANS LA POSITION
Y DU CARACTERE 0
530 GOTO340
600 REM ***** DONNEES DES CARACTERES GRAPHIQUES PROGRAMMABLES
*****
610 DATA0,127,0,1,255,192,3,255,224,3,231,224
620 DATA7,217,240,7,223,240,7,217,240,3,231,224
630 DATA3,255,224,3,255,224,2,255,160,1,127,64
640 DATA1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
650 DATA0,62,0,0,62,0,0,62,0,0,28,0,0
```

```
10 REM EXEMPLE 2 DE CARACTERES GRAPHIQUES PROGRAMMABLES . . .
20 REM TOUJOURS LE BALLON A AIR CHAUD
30 VIC=13*4096:REM LES REGISTRES VIC COMMENCENT ICI
35 POKEVIC+21,63:REM VALIDE LES CARACTERES 0 A 5
36 POKEVIC+33,14:REM FIXE LA COULEUR D'ARRIERE-PLAN AU BLEU CLAIR
37 POKEVIC+23,3:REM ETALE LES CARACTERES 0 ET 1 DANS LE SENS Y
38 POKEVIC+29,3:REM ETALE LES CARACTERES 0 ET 1 DANS LE SENS X
40 POKE2040,192:REM FIXE LE POINTEUR DU CARACTERE 0
50 POKE2041,193:REM FIXE LE POINTEUR DU CARACTERE 1
60 POKE2042,192:REM FIXE LE POINTEUR DU CARACTERE 2
70 POKE2043,193:REM FIXE LE POINTEUR DU CARACTERE 3
80 POKE2044,192:REM FIXE LE POINTEUR DU CARACTERE 4
90 POKE2045,193:REM FIXE LE POINTEUR DU CARACTERE 5
100 POKEVIC+4,30:REM FIXE LA POSITION X DU CARACTERE 2
110 POKEVIC+5,58:REM FIXE LA POSITION Y DU CARACTERE 2
120 POKEVIC+6,65:REM FIXE LA POSITION X DU CARACTERE 3
130 POKEVIC+7,58:REM FIXE LA POSITION Y DU CARACTERE 3
140 POKEVIC+8,100:REM FIXE LA POSITION X DU CARACTERE 4
150 POKEVIC+9,58:REM FIXE LA POSITION Y DU CARACTERE 4
160 POKEVIC+10,100:REM FIXE LA POSITION X DU CARACTERE 5
170 POKEVIC + 11,58:REM FIXE LA POSITION Y DU CARACTERE 5
           CTRL 2
175 PRINT"[graphic]" TAB(15)"DEUX CARACTERES GRAPHIQUES PROGRAM-
MABLES"     SHIFT CLR/HOME
176 PRINTTAB(55)"L'UN SUR L'AUTRE"
180 POKEVIC + 0,100:REM FIXE LA POSITION X DU CARACTERE 0
190 POKEVIC+1,100:REM FIXE LA POSITION Y DU CARACTERE 0
200 POKEVIC+2,100:REM FIXE LA POSITION X DU CARACTERE 1
```

```
210 POKEVIC+3,100:REM FIXE LA POSITION Y DU CARACTERE 1
220 POKEVIC+39,1:REM FIXE LA COULEUR DU CARACTERE 0
230 POKEVIC+41,1:REM FIXE LA COULEUR DU CARACTERE 2
240 POKEVIC+43,1:REM FIXE LA COULEUR DU CARACTERE 4
250 POKEVIC+40,6:REM FIXE LA COULEUR DU CARACTERE 1
260 POKEVIC+42,6:REM FIXE LA COULEUR DU CARACTERE 3
270 POKEVIC+44,6:REM FIXE LA COULEUR DU CARACTERE 5
280 FORX=192TO193:REM DEBUT DE LA BOUCLE QUI DEFINIT LES CARACTERES
GRAPHIQUES PROGRAMMABLES
290 FORY=0TO63:REM COMPTEUR D'OCTET AVEC BOUCLE DE CARACTERE
300 READA:REM LECTURE DANS UN OCTET
310 POKEX*64+Y,A:REM STOCKE LES DONNEES DANS LA ZONE DE CARACTERE
GRAPHIQUE
320 NEXTY,X:REM FERMETURE DES BOUCLES
330 DX=1:DY=1
340 X=PEEK(VIC):REM OBSERVE LA POSITION X DU CARACTERE 0
350 Y=PEEK(VIC+1):REM OBSERVE LA POSITION Y DU CARACTERE 0
360 IFY=50ORY=208THENDY=-DY:REM SI Y EST AU BORD DE . . .
370 REM L'ECRAN, ALORS DELTA Y INVERSE
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SI LE CARACTERE
GRAPHIQUE PROGRAMMABLE . . .
390 REM TOUCHE LE BORD GAUCHE, L'INVERSER
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SI LE CARACTERE
PROGRAMMABLE . . .
410 REM TOUCHE LE BORD DROIT, L'INVERSER
420 IFX=255ANDDX=1THENX=-1:SIDE=3
430 REM PASSER A L'AUTRE COTE DE L'ECRAN
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM PASSER A L'AUTRE COTE DE L'ECRAN
460 X=X+DX:REM AJOUTER DELTA X A X
470 X=XAND255:REM S'ASSURER QUE X EST DANS L'INTERVALLE PERMIS
480 Y=Y+DY:REM AJOUTER DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM METTRE LA NOUVELLE VALEUR X DANS LA POSITION X DU
CARACTERE 0
500 POKEVIC+2,X:REM METTRE LA NOUVELLE VALEUR X DANS LA POSITION X
DU CARACTERE 1
510 POKEVIC+1,Y:REM METTRE LA NOUVELLE VALEUR Y DANS LA POSITION Y DU
CARACTERE 0
```

```
520 POKEVIC+3,Y:REM METTRE LA NOUVELLE VALEUR Y DANS LA POSITION Y
DU CARACTERE 1
530 GOTO340
600 REM ***** DONNEES DES CARACTERES GRAPHIQUES PROGRAMMABLES
*****
610 DATA0,255,0,3,153,192,7,24,224,7,56,224,14,126,112,14,126,112,14,126, 112
620 DATA6,126,96,7,56,224,7,56,224,1,56,128,0,153,0,0,90,0,0,56, 0
630 DATA0,56,0,0,0,0,0,0,0,0,126,0,0,42,0,0,84,0, 0,40,0,0
640 DATA0,0,0,0,102,0,0,231,0,0,195,0,1,129,128,1,129, 128,1,129,128
650 DATA1,129,128,0,195,0,0,195,0,4,195,32,2,102,64,2,36,64,1,0, 128
660 DATA1,0,128,0,153,0,0,153,0,0,0,0,0,84,0,0,42,0,0, 20,0,0
```

```
10 REM EXEMPLE 3 DE CARACTERES GRAPHIQUES PROGRAMMABLES . . .
20 REM LA BOULE D'AIR CHAUD
30 VIC=53248:REM LES REGISTRES VIC COMMENCENT ICI
35 POKEVIC+21,1:REM VALIDATION DU CARACTERE GRAPHIQUE 0
36 POKEVIC+33,14:REM FIXE LA COULEUR D'ARRIERE-PLAN AU BLEU CLAIR
37 POKEVIC+23,1:REM ETALE LE CARACTERE 0 DANS LE SENS Y
38 POKEVIC+29,1:REM ETALE LE CARACTERE 0 DANS LE SENS X
40 POKE2040,192:REM FIXE LE POINTEUR DU CARACTERE 0
50 POKEVIC+28,1:REM PASSE EN MODE MULTICOLORE
60 POKEVIC+37,7:REM FIXE MULTICOLORE 0
70 POKEVIC+38,4:REM FIXE MULTICOLORE 1
180 POKEVIC+ 0,100:REM FIXE LA POSITION X DU CARACTERE 0
190 POKEVIC+1,100:REM FIXE LA POSITION Y DU CARACTERE 0
220 POKEVIC+39,2:REM FIXE LA COULEUR DU CARACTERE 0
290 FORY= 0TO63:REM COMPTEUR D'OCTET AVEC BOUCLE DE CARACTERE
300 READA:REM LECTURE DANS UN OCTET
```

```
310 POKE12288+Y,A:REM STOCKE LES DONNEES DANS LA ZONE DE CARACTERE
320 NEXT Y:REM FERME LA BOUCLE
330 DX = 1:DY = 1
340 X = PEEK(VIC):REM OBSERVE LA POSITION X DU CARACTERE 0
350 Y=PEEK(VIC+1):REM OBSERVE LA POSITION Y DU CARACTERE 0
360 IFY=50ORY=208THENDY=-DY:REM SI Y EST AU BORD DE . . .
370 REM L'ECRAN, INVERSER ALORS DELTA Y
380 IF X=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SI LE CARACTERE
. . .
390 REM TOUCHE LE BORD GAUCHE, L'INVERSER
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SI LE CARACTERE . . .
410 REM TOUCHE LE BORD DROIT, L'INVERSER
420 IFX=255ANDDX=1THENX=-1:SIDE=1
430 REM PASSER A L'AUTRE COTE DE L'ECRAN
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM PASSER A L'AUTRE COTE DE L'ECRAN
460 X = X + DX:REM AJOUTER DELTA X A X
470 X = XAND255:REM S'ASSURER QUE X EST DANS L'INTERVALLE PERMIS
480 Y=Y+DY:REM AJOUTER DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM MET LA NOUVELLE VALEUR X DANS LA POSITION X DU
CARACTERE 0
510 POKEVIC+1,Y:REM MET LA NOUVELLE VALEUR Y DANS LA POSITION Y DU
CARACTERE 0
520 GETA$:REM INTRODUIT UNE TOUCHE DU CLAVIER
521 IFA$="M"THENPOKEVIC+28,1:REM MULTICOLORE CHOISI PAR L'UTILISATEUR
522 IFA$="H"THENPOKEVIC+28,0:REM HAUTE DEFINITION CHOISIE PAR
L'UTILISATEUR
530 GOTO340
600 REM ***** DONNEES DES CARACTERES GRAPHIQUES PROGRAMMABLES
*****
610 DATA64,0,1,16,170,4,6,170,144,10,170,160,42,170,168,41, 105,104, 169 ,235,106
620 DATA169,235,106,169,235,106,170,170,170,170,170,170, 170,170, 170,170, 170,170
630 DATA166,170,154,169,85,106,170,85,170,42,170,168,10,170, 160,1,0,64,1,0,64
640 DATA5,0,80,0
```

AUTRES CARACTÉRISTIQUES GRAPHIQUES

EFFACEMENT DE L'ÉCRAN

Le bit 4 du registre de commande VIC-II commande la fonction d'effacement de l'écran. Il se trouve dans le registre de commande, à la position 53265 ($D011). Quand il est à 1 (en fonction), l'écran est normal. Quand le bit 4 passe à 0 (hors fonction), l'écran entier passe à la couleur de son cadre.

L'instruction POKE suivante efface l'écran. Les données ne sont pas perdues; elles ne sont simplement pas affichées.

POKE 53265,PEEK(53265)AND 239

Pour faire réapparaître l'écran, utiliser l'instruction POKE suivante:

POKE 53265,PEEK(53265)OR 16

REMARQUE: L'unité de traitement accélère légèrement si on coupe l'écran. L'exécution (RUN) du programme se fait donc aussi plus vite.

REGISTRE DE TRAME

Le registre de trame, qui a deux fonctions, se trouve dans la microplaquette VIC-II à la position 53266 ($D012). Quand on lit ce registre, il donne les 8 bits inférieurs de la position de trame présente. La position de trame du bit le plus significatif est à la position 53265 ($D011) du registre. On utilise le registre de trame pour fixer les changements de synchronisation de l'affichage afin d'éliminer le scintillement de l'écran. Faire les changements sur l'écran quand la trame n'est pas dans la zone d'affichage visible, c'est-à-dire quand les positions de points se situent entre 51 et 251.

Quand on écrit dans le registre de trame (y compris le bit le plus significatif), le nombre écrit est sauvegardé pour l'utilisation avec la fonction de comparaison de trame. Quand la valeur réelle de trame devient identique au nombre écrit dans le registre de trame, un bit du registre d'interruption de microplaquette VIC-II à la position 53273 ($D019) passe à 1 (en fonction).

REMARQUE: Si le bit correct d'interruption est validé (mis en fonction), il se produit une interruption (RQ).

REGISTRE D'ÉTAT D'INTERRUPTION

Le registre d'état d'interruption indique l'état courant de toute source d'interruption. L'état courant du bit 2 du registre d'interruption est à 1 quand deux caractères graphiques programmables entrent en collision. Il en est de même, en rapport direct, pour les bits 0 à 3 indiqués dans le tableau ci-dessous. Le bit 7 est également mis à 1 quand il se produit une interruption.

Le registre d'état d'interruption est à la position 53273 ($D019) et ses conditions sont les suivantes:$

VERROU	BIT	DESCRIPTION
IRST	0	Fixé si compte de trame courant = compte de trame stocké
IMDC	1	Fixé par collision caractère graphique/donnée (1re seulement, jusqu'à remise à l'état initial)
IMMC	2	Fixé par collision caractère graphique programmable/ caractère graphique programmable (1re seulement, jusqu'à remise à l'état initial)
ILP	3	Fixé par transition négative du crayon lumineux (1 par cadre)
IRQ	7	Fixé par verrou et validé

Quand un bit d'interruption est fixé, il est "verrouillé" et doit être effacé par l'écriture d'un 1 à ce bit dans le registre d'interruption, quand on est prêt à le traiter. On dispose ainsi d'un traitement sélectif des interruptions, sans devoir stocker les autres bits d'interruption.

Le **REGISTRE DE VALIDATION D'INTERRUPTION** est à la position 53274 ($D01A). Son format est le même que celui du registre d'état d'interruption. Si le bit correspondant du registre de validation d'interruption n'est pas fixé à 1, il ne se produit aucune interruption à partir de cette source. On peut encore interroger le registre d'état d'interruption, mais il n'est créé aucune interruption.

Pour valider une demande d'interruption, le bit de validation d'interruption correspondant (voir tableau ci-dessus) doit est être fixé à 1.

Cette puissante structure d'interruption permet d'utiliser des modes d'écran divisé. Par exemple, on peut avoir une moitié d'écran en topographie binaire, l'autre en texte, plus de 8 caractères graphiques programmables à la fois, etc. Il faut simplement savoir utiliser convenablement les interruptions. Par exemple, si l'on désire avoir la moitié supérieure de l'écran en topographie binaire et la moitié inférieure en texte, fixer le registre de comparaison de trame (expliqué précédemment) pour la partie inférieure de l'écran. Quand l'interruption se produit, indiquer à la microplaquette VIC-II de sortir des caractères de la mémoire morte ROM, puis fixer le registre de comparaison de trame pour l'interruption en haut de l'écran. Quand l'interruption se produit en haut de l'écran, indiquer à la VIC-II de sortir des caractères de la mémoire vive RAM (mode de topographie binaire).

On peut aussi afficher plus de 8 caractères graphiques programmables de la même manière. Le BASIC n'est malheureusement pas assez rapide pour donner de bons résultats. Si l'on désire donc commencer à utiliser des interruptions d'affichage, il faut travailler en langage machine.

COMBINAISONS SUGGÉRÉES DE COULEURS D'ÉCRAN ET DE CARACTÈRES

Les possibilités des télécouleurs sont limitées pour la combinaison de certaines couleurs sur la même ligne. Certaines combinaisons de couleur d'écran et de caractères donnent des images floues. Le tableau ci-dessous indique les combinaisons de couleurs à éviter et celles qui donnent d'excellents résultats.

COULEUR DE CARACTÈRE (colonnes) / COULEUR D'ÉCRAN (lignes)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	●	X	●	●	•	X	●	●	X	●	●	●	●	●	●
1	●	X	●	X	●	●	●	X	•	●	•	●	●	X	●	●
2	X	●	X	X	•	X	X	●	●	X	●	X	X	X	X	•
3	●	X	X	X	X	•	●	X	X	X	X	•	X	X	•	X
4	●	•	X	X	X	X	X	X	X	X	X	X	X	X	X	•
5	●	•	X	•	X	X	X	X	X	X	X	•	X	●	X	•
6	•	●	X	●	X	X	X	X	X	X	X	X	X	•	●	●
7	●	X	●	X	X	X	•	X	•	●	•	●	●	X	X	X
8	•	●	●	X	X	X	X	●	X	●	X	X	X	X	X	•
9	X	●	X	X	X	X	X	●	●	X	●	X	X	X	X	●
10	•	•	●	X	X	X	X	•	X	●	X	X	X	X	X	•
11	●	●	X	•	X	X	X	●	X	X	X	X	●	●	•	●
12	●	●	•	X	X	X	•	X	X	•	X	●	X	X	X	●
13	●	X	X	X	X	●	•	X	X	X	X	●	X	X	X	X
14	●	●	X	●	X	X	●	X	X	X	X	•	X	X	X	•
15	●	●	●	X	•	•	●	X	X	•	•	●	●	X	•	X

● = Excellent
• = Bon
X = Mauvais

AUTRE MÉTHODE DE PROGRAMMATION DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

Si l'on rencontre des difficultés avec les caractères graphiques, la présente section offre une méthode d'enseignement plus simple des caractères graphiques programmables.

PRÉPARATION DES CARACTÈRES GRAPHIQUES PROGRAMMABLES EN BASIC—PROGRAMME ABRÉGÉ

Il existe au moins trois techniques de programmation BASIC différentes qui permettent de créer des images graphiques et des animations de type dessin animé avec le Commodore 64. On peut utiliser le jeu incorporé de caractères graphiques de l'ordinateur (voir page 376). L'utilisateur peut programmer ses propres caractères (voir page 108) ou (et cela est préférable), il peut utiliser les caractères graphiques programmables intégrés de l'ordinateur. Pour en montrer la facilité, voici l'un des programmes les plus courts qu'il soit possible d'écrire en BASIC pour la création de caractères graphiques programmables.

SHIFT CLR/HOME

```
10 PRINT" ♥"
20 POKE2040,13
30 FORS=832TO832+62:POKES,255:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV,24
80 POKEV+1,100
```

Ce programme comprend les éléments clés nécessaires à la création d'un caractère graphique programmable. Les nombres POKE viennent du tableau de préparation des caractères graphiques programmables de la page 176. Ce programme définit le premier caractère graphique programmable ou caractère 0 comme un carré blanc plein sur l'écran. Nous donnons ci-dessous une explication détaillée de ce programme:

LA LIGNE 10 efface l'écran.

LA LIGNE 20 place le pointeur de caractère graphique programmable où le Commodore doit lire ses données de caractère graphique programmable. Le caractère 0 est à 2040, le caractère 1 à 2041, le caractère 2 à 2042 et ainsi de suite jusqu'au caractère 7 qui est à 2047. On peut fixer les 8 pointeurs de caractère graphique programmable à 13 en utilisant la ligne suivante à la place de la ligne 20:

20 FOR SP=2040TO2047:POKE SP,13:NEXT SP

LA LIGNE 30 met le premier caractère graphique programmable (caractère 0) dans 63 octets de la mémoire vive RAM du Commodore 64, à partir de la position 832 (chaque caractère graphique programmable occupe 63 octets de mémoire). Le premier caractère graphique programmable (caractère 0) est "adressé" aux positions de mémoire 832 à 894.

LA LIGNE 40 fixe la variable "V" égale à 53248 qui est l'adresse de départ de la microplaquette vidéo. Cette entrée nous permet d'utiliser la forme (V+nombre) pour les réglages de caractère graphique programmable. Nous utilisons la forme (V+nombre) pour écrire (POKE) les réglages de caractère graphique programmable. Ce format conserve de la mémoire et permet de travailler avec des nombres plus petits. Par exemple, à la ligne 50, nous avons tapé POKE V+21; cette expression équivaut à POKE 53248+21 ou POKE 53269; V+21, qui est plus facile à mémoriser, prend aussi moins de place que 53269.

LA LIGNE 50 valide ou met en fonction le caractère graphique programmable 0. Nous disposons de 8 caractères graphiques programmables numérotés de 0 à 7. Pour mettre en fonction un caractère graphique programmable séparé ou une combinaison de ces caractères, il suffit de taper POKE V+21 suivi d'un nombre de 0 (tous les caractères graphiques programmables sont coupés) à 255 (les 8 caractères graphiques programmables sont en fonction). On peut mettre en fonction un ou plusieurs caractères graphiques programmables en écrivant (POKE) les nombres suivants:

TOUS EN FONCTION	CARACT. 0	CARACT. 1	CARACT. 2	CARACT. 3	CARACT. 4	CARACT. 5	CARACT. 6	CARACT. 7	TOUS COUPÉS
V+21 255	V+21 1	V+21 2	V+21 4	V+21 8	V+21 16	V+21 32	V+ 21 64	V+21 128	V+21 0

POKE V+21,1 met en fonction le caractère 0. V+21,128 met le caractère 7 en fonction. On peut aussi mettre en fonction des combinaisons de caractères graphiques programmables. Par exemple, POKE V+21,129 met en fonction le caractère 0 et le caractère 7 en ajoutant les deux nombres de "mise en fonction" (1+128). (Voir le tableau de préparation des caractères graphiques programmables à la page 176.)

LA LIGNE 60 fixe la couleur du caractère graphique 0. Il existe 16 couleurs possibles de caractère graphique programmable numérotées de 0 (noir) à 15 (gris). Pour fixer la couleur de chaque caractère graphique programmable, il faut une instruction POKE différente, de V+39 à V+46. Avec POKE V+39,1, le caractère 0 est blanc. Avec POKE V+46,15, le caractère 7 est gris. (Pour plus de détails, voir le tableau de préparation des caractères graphiques programmables.)

Quand on crée un caractère graphique programmable comme on vient de le faire, il reste en mémoire jusqu'à ce qu'on le coupe avec une instruction POKE, qu'on le redéfinisse ou qu'on arrête l'ordinateur. On peut ainsi changer la couleur, la position et même

la forme du caractère graphique programmable en mode direct ou immédiat, solution très pratique pour l'édition. À titre d'exemple, exécuter (RUN) le programme ci-dessus, puis taper la ligne suivante en mode direct (sans numéro de ligne) et appuyer sur la touche RETURN :

POKE V+39,8

Le caractère graphique programmable sur l'écran est maintenant orange. Essayer d'écrire (POKE) d'autres nombres de 0 à 15 pour faire apparaître les autres couleurs des caractères graphiques programmables. Comme on a utilisé le mode direct, le caractère revient à sa couleur initiale (blanc) si l'on exécute (RUN) le programme.

LA LIGNE 70 détermine la position horizontale ou "X" du caractère graphique programmable sur l'écran. Ce nombre représente la position du coin supérieur gauche du caractère graphique programmable. La position horizontale (X) la plus à gauche que l'on puisse voir sur l'écran est la position numéro 24, mais on peut déplacer le caractère en dehors de l'écran, à la position numéro 0.

LA LIGNE 80 détermine la position verticale ou "Y" du caractère graphique programmable. Dans ce programme, nous avons mis le caractère graphique à la position horizontale (X) 24 et à la position verticale (Y) 100. Pour passer à une autre position, essayer l'instruction POKE suivante en mode direct et appuyer sur RETURN :

POKE V,24:POKE V+1,50

On a placé le caractère graphique programmable dans le coin supérieur gauche de l'écran. Pour amener le caractère dans le coin inférieur gauche, taper la ligne suivante:

POKE V,24:POKE V+1,229

Dans l'adresse du caractère graphique programmable 0, chaque nombre de 832 à 895 représente un bloc de 8 points d'image; il y a trois blocs de 8 éléments d'image dans chaque rangée horizontale du caractère graphique programmable. La boucle de la ligne 80 indique à l'ordinateur d'écrire (POKE) 832,255 qui remplit les 8 premiers éléments d'image; ensuite, POKE 833,255 remplit les 8 éléments d'image suivants. On continue ainsi jusqu'à la position 894 qui correspond au dernier groupe de 8 éléments d'image, dans le coin inférieur droit du caractère graphique programmable. Pour mieux comprendre ce processus, taper la ligne suivante en mode direct; on peut remarquer que le deuxième groupe de 8 éléments d'image est effacé:

POKE 833,0 (pour le faire réapparaître, taper POKE 833,255 ou exécuter le programme)

La ligne suivante, que l'on peut ajouter au programme, efface les blocs au milieu du caractère graphique programmable que l'on a créé:

90 FOR A=836 TO 891 STEP 3:POKE A,0:NEXT A

Ne pas oublier que les éléments d'image qui constituent le caractère graphique programmable sont groupés par blocs de 8. Cette ligne efface le 5e groupe de 8 éléments d'images (bloc 836) et un bloc sur trois jusqu'à 890. Essayer d'écrire (POKE) l'un des autres nombres de 832 à 894 avec un nombre 255 pour les remplir ou avec un 0 pour les effacer.

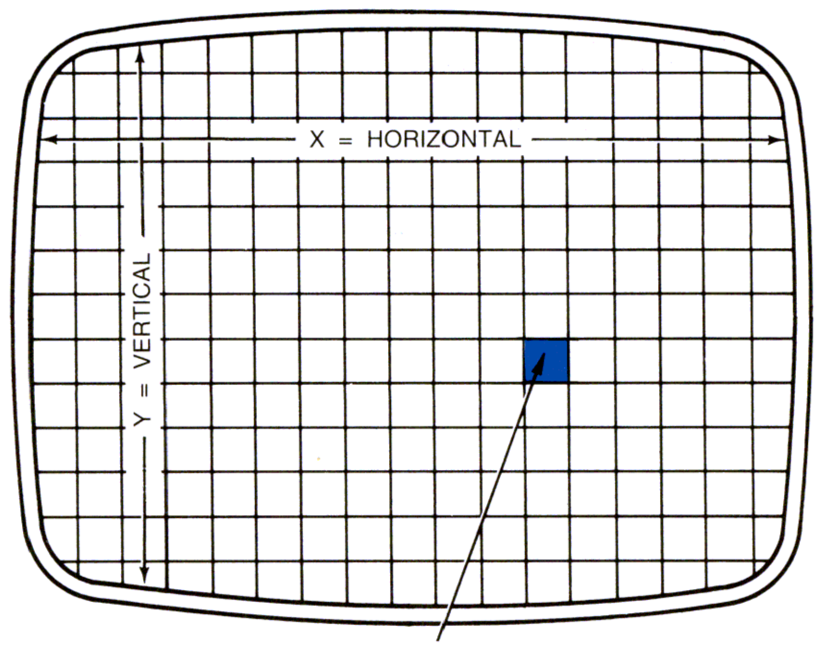
COMPRESSION DES PROGRAMMES DE CARACTÈRES GRAPHIQUES PROGRAMMABLES

Conseil pratique de compression: le programme ci-dessus est déjà court, mais on peut encore l'abréger en le "compressant". Dans notre exemple, nous avons indiqué les réglages principaux de caractère graphique programmable sur des lignes séparées pour que l'on puisse comprendre le déroulement du programme. Dans la réalité, un bon programmeur ramènerait probablement ce programme à deux lignes en le comprimant de la façon suivante:

```
10PRINTCHR$(147):V=53248:POKEV+21,1:POKE2040,13:
   POKEV+39,1
20FORS=832TO894:POKES,255:NEXT:POKEV,24:POKEV+1,100
```

Pour plus de détails sur la compression des programmes afin de leur faire occuper moins de mémoire et de les exécuter plus efficacement, voir le "guide de compression" à la page 24.





On doit fixer la position X (horizontale) et la position Y (verticale) d'un caractère graphique programmable à ce point pour pouvoir l'afficher sur l'écran.

Figure 3-4. L'écran d'affichage est divisé en une grille de coordonnées X et Y.

POSITIONNEMENT DES CARACTÈRES GRAPHIQUES PROGRAMMABLES SUR L'ÉCRAN

L'écran de visualisation complet est divisé en une grille de coordonnées X et Y, comme un graphique. La coordonnée X correspond à la position horizontale sur l'écran et la coordonnée Y à la position verticale (voir figure 3-4).

Pour placer un caractère graphique programmable sur l'écran, on doit écrire (POKE) deux réglages qui sont la position X et la position Y pour indiquer à l'ordinateur où afficher le coin supérieur gauche du caractère. Il ne faut pas oublier qu'un caractère graphique programmable se compose de 504 éléments d'image individuels (24 dans le sens horizontal par 21 dans le sens vertical). Si l'on inscrit (POKE) donc un caractère graphique

programmable dans le coin supérieur gauche de l'écran, ce caractère est affiché sous forme d'image graphique de 24 éléments d'image horizontaux par 21 éléments d'image verticaux, à partir des positions X et Y que l'on a définies. Le caractère graphique programmable est affiché en fonction de son coin supérieur gauche, même si on le définit en n'utilisant qu'une petite partie de la zone de 24 par 21 éléments d'image.

Pour comprendre le positionnement dans les sens X et Y, étudier la figure 3-5 qui montre les nombres X et Y par rapport à l'écran de visualisation. Remarquer que la zone grise correspond à la partie visible de l'écran du téléviseur; la zone blanche représente les positions qui ne sont pas visibles.

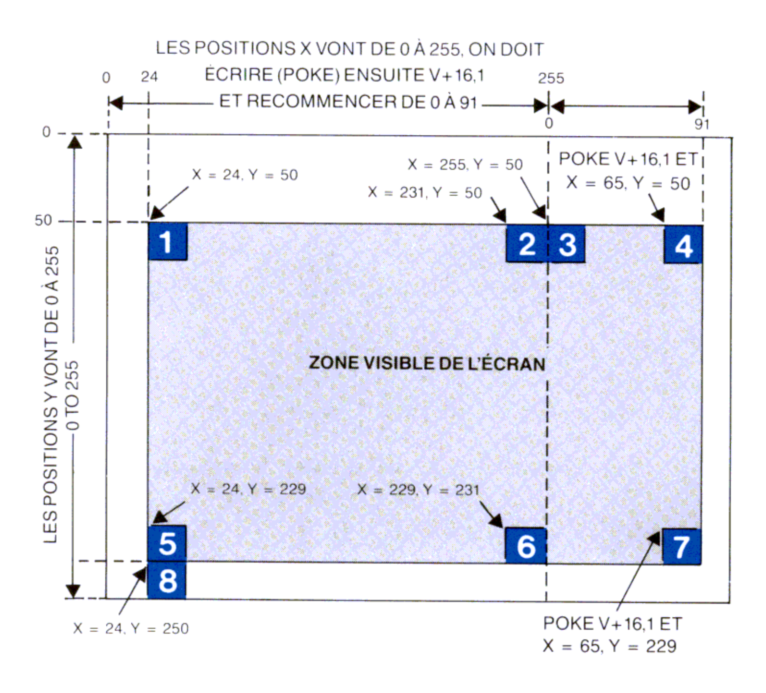


Figure 3-5. Détermination des positions X et Y de caractère graphique programmable.

Pour afficher un caractère graphique programmable à une position donnée, on doit écrire (POKE) les réglages X et Y de chaque caractère en se rappelant que chacun d'eux possède ses propres instructions POKE X et Y uniques. Les réglages X et Y des 8 caractères graphiques programmables sont indiqués ci-dessous:

ÉCRIRE (POKE) LES VALEURS SUIVANTES POUR FIXER LES POSITIONS X ET Y

	CARACT.0	CARACT.1	CARACT.2	CARACT.3	CARACT.4	CARACT.5	CARACT.6	CARACT.7
VALEUR X	V,X	V+2,X	V+4,X	V+6,X	V+8X	V+10,X	V+12,X	V+14,X
VALEUR Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
X DE DROITE	V+16,1	V+16,2	V+16,4	V+16,8	V+16,16	V+16,32	V+16,64	V+16,128

ÉCRITURE (POKE) D'UNE POSITION X: Les valeurs possibles de X vont de 0 à 255, en allant de la gauche vers la droite. Les valeurs 0 à 23 placent un caractère graphique programmable partiellement ou en totalité hors de la zone visible, à la gauche de l'écran; les valeurs 24 à 255 placent le caractère graphique dans la zone visible, jusqu'à la 255e position (voir le paragraphe suivant pour les réglages après la 255e position X). Pour placer le caractère graphique programmable à l'une de ces positions, taper l'instruction POKE de position X pour le caractère utilisé. Par exemple, pour mettre le caractère graphique 1 à la position X d'extrême gauche, dans la zone visible, taper POKE V+2,24.

VALEURS X AU-DELÀ DE LA 255e POSITION: Pour aller au-delà de la 255e position sur l'écran, on doit utiliser une deuxième instruction POKE avec les nombres de la rangée X de droite du tableau (figure 3-5). La numérotation horizontale (X) continue normalement après la 255e position jusqu'à 256, 257, etc., mais les registres ne contenant que 8 bits, nous devons en utiliser un deuxième pour accéder au côté droit de l'écran et recommencer la numérotation X à 0. Pour aller au-delà de la position X 255, on doit donc utiliser une instruction POKE V+16 et un nombre (fonction du caractère graphique programmable). On dispose ainsi de 65 autres positions X (renumérotées de 0 à 65) dans la zone visible de la droite de l'écran. (On peut en fait utiliser une instruction POKE avec une valeur X de droite atteignant 255 pour sortir du bord de droite de l'écran.)

ÉCRITURE (POKE) D'UNE POSITION Y: Les valeurs possibles de Y vont de 0 à 255, du haut vers le bas. Les valeurs 0 à 49 mettent le caractère graphique programmable totalement ou partiellement en dehors de la zone visible, en haut de l'écran. Les valeurs 50 à 229 mettent le caractère graphique programmable dans la zone visible. Les valeurs 230 à 255 mettent le caractère graphique programmable totalement ou partiellement en dehors de la zone visible, au bas de l'écran.

Étudions le positionnement X-Y avec le caractère graphique programmable 1. Essayer le programme suivant:

```
         SHIFT CLR/HOME
10 PRINT "⌂":V=53248:POKEV+21,2:POKE2041,13:
FORS=832TO895:POKES,255:NEXT
20 POKEV+40,7
30 POKEV+2,24
40 POKEV+3,50
```

Ce programme simple établit le caractère graphique 1 sous la forme d'une case pleine et le place dans le coin supérieur gauche de l'écran. Changer maintenant la ligne 40 pour avoir:

40 POKE V+3,229

On déplace ainsi le caractère graphique programmable dans le coin inférieur gauche de l'écran. Essayons maintenant la limite X de droite de caractère graphique programmable. Changer la ligne 30 pour qu'on ait:

30 POKE V+2,255

On déplace ainsi le caractère graphique programmable vers la droite et on atteint la limite X de droite qui est 255. À ce point, le bit le plus significatif dans le registre 16 doit être fixé. On doit donc taper POKE V+16 et le nombre indiqué dans la colonne "de droite" dans le tableau d'instructions POKE X-Y ci-dessus pour remettre le compteur de position X à la 256ᵉ position d'élément d'image sur l'écran. Changer la ligne 30 pour qu'on lise:

30 POKE V+16, PEEK(V+16)OR 2:POKE V+2,0

POKE V+16,2 fixe le bit le plus significatif de la position X pour le caractère graphique programmable 1 et le remet à la 256ᵉ position d'élément d'image sur l'écran. **POKE V+2,0** affiche le caractère graphique programmable à la nouvelle position zéro qui est maintenant remise au 256ᵉ élément d'image.

Pour revenir au côté gauche de l'écran, on doit remettre le bit le plus significatif du compteur de position X à 0 en tapant (pour le caractère graphique programmable 1):

POKE V+16, PEEK(V+16)AND 253

POUR RÉSUMER le fonctionnement du positionnement X . . . écrire (POKE) la position X d'un caractère graphique programmable avec un nombre de 0 à 255. Pour accéder à une position au-delà de la 255ᵉ position d'élément d'image sur l'écran, on doit ajouter une autre instruction POKE (V+16) qui fixe le bit le plus significatif de la position X et recommence le comptage à 0, au 256ᵉ élément d'image sur l'écran.

Cette instruction POKE recommence la numérotation X à 0 à la 256e position. **(Exemple:** on doit inclure **POKE V+16, PEEK(V+16)OR 1** et **POKE V,1** pour mettre le caractère graphique programmable 0 au 257e élément d'image sur l'écran.) Pour revenir à gauche de X positions, on doit couper le réglage de commande en tapant **POKE V+16, PEEK(V+16)AND 254.**

POSITIONNEMENT DE PLUSIEURS CARACTÈRES GRAPHIQUES PROGRAMMABLES

Le programme suivant définit trois caractères graphiques programmables différents (0, 1 et 2) et de couleurs différentes; ils sont aussi placés à des positions différentes sur l'écran:

SHIFT CLR/HOME

```
10 PRINT"  ":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50
60 POKEV+2,12:POKEV+3,229
70 POKEV+4,255:POKEV+5,50
```

Pour faciliter les choses, les 3 caractères graphiques programmables sont définis sous forme de carrés pleins et on extrait leurs données du même endroit. Il faut ici surtout s'intéresser au positionnement de ces 3 caractères. Le caractère graphique programmable blanc 0 est dans le coin supérieur gauche. Le caractère graphique programmable jaune 1 est dans le coin inférieur gauche, mais une moitié est en dehors de l'écran (24 est la position X d'extrême gauche de la zone visible; une position X inférieure à 24 place donc le caractère graphique programmable totalement ou partiellement hors de l'écran; nous avons utilisé ici une position X 12 qui met donc une moitié de caractère en dehors de l'écran). Enfin, le caractère graphique programmable orange 2 est à la limite X de droite (position 255). Que se passe-t-il donc si l'on veut afficher un caractère graphique programmable dans la zone à droite de la position X 255?

AFFICHAGE D'UN CARACTÈRE GRAPHIQUE PROGRAMMABLE AU-DELÀ DE LA 255e POSITION X

L'affichage d'un caractère graphique programmable au-delà de la 255e position X se fait avec une instruction POKE spéciale qui fixe le bit le plus significatif de la position X et commence à la 256e position d'élément d'image sur l'écran. En voici le déroulement . . .

Taper d'abord POKE V+16 avec le numéro du caractère graphique programmable utilisé (nous utilisons ici le caractère graphique programmable 0; voir la rangée X de "droite"

dans le tableau X-Y). On attribue maintenant une position X, sans oublier que le compteur X recommence à 0 à la 256e position sur l'écran. Changer la ligne 50 pour qu'on lise:

50 POKE V+16,1:POKE V,24:POKE V+1,75

Cette ligne utilise POKE V+16 avec le nombre requis pour "ouvrir" le côté droit de l'écran; la nouvelle position X 24 pour le caractère graphique programmable 0 commence maintenant à 24 éléments d'image à la droite de la position 255. Pour vérifier le bord de droite de l'écran, changer la ligne 60 pour qu'on ait:

60 POKE V+16,1:POKE V,65:POKE V+1,75

Quelques essais avec les réglages du tableau de caractères graphiques programmables permettent de positionner et de déplacer les caractères graphiques programmables sur les côtés de gauche et de droite de l'écran. La section "Déplacement du caractère graphique programmable" permet aussi de mieux comprendre le positionnement de ces caractères.

PRIORITÉS DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

On peut donner réellement l'impression que différents caractères graphiques programmables se déplacent devant ou derrière les uns les autres sur l'écran. On arrive à cet incroyable effet tridimensionnel avec les priorités des caractères graphiques programmables qui déterminent les caractères prioritaires par rapport aux autres quand 2 ou plusieurs d'entre eux se recouvrent sur l'écran.

L'application de la règle "premier entré, premier traité" donne automatiquement la priorité aux caractères graphiques programmables de numéro plus bas. Par exemple, si l'on affiche les caractères 0 et 1 de façon qu'ils se recouvrent sur l'écran, le caractère 0 paraît être devant le caractère 1. Le caractère 0 a en fait la priorité sur tous les autres parce qu'il possède le numéro le plus bas. Le caractère 1 a ainsi la priorité sur les caractères 2 à 7; le caractère 2 a la priorité sur les caractères 3 à 7, etc. Le caractère 7 (dernier caractère graphique programmable) a une priorité inférieure à celle de tous les autres caractères; il paraît toujours être affiché derrière les autres caractères qui recouvrent sa position.

Pour illustrer le processus des priorités, changer les lignes 50, 60 et 70 du programme ci-dessus pour qu'on ait:

```
          SHIFT CLR/HOME
10 PRINT "♥":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50:PODEV+16,0
60 POKEV+2,34:PODEV+3,60
70 POKEV+4,44:POKEV+5,70
```

On doit voir un caractère graphique programmable blanc par-dessus un caractère jaune, lequel recouvre un caractère orange. Maintenant que l'on a compris le processus des priorités, on peut aussi déplacer les caractères graphiques programmables et tirer parti des priorités dans l'animation.

DESSIN D'UN CARACTÈRE GRAPHIQUE PROGRAMMABLE

Le dessin d'un caractère graphique programmable Commodore revient à couvrir les espaces vides de couleur, comme dans un livre à colorier. Chaque caractère graphique programmable se compose de points minuscules ou éléments d'image. Pour dessiner un caractère graphique programmable, il suffit de "colorier" certains des éléments d'image.

Jetons un coup d'oeil à la grille de dessin de caractère graphique programmable de la figure 3-6. Un caractère graphique programmable vierge a l'aspect suivant:

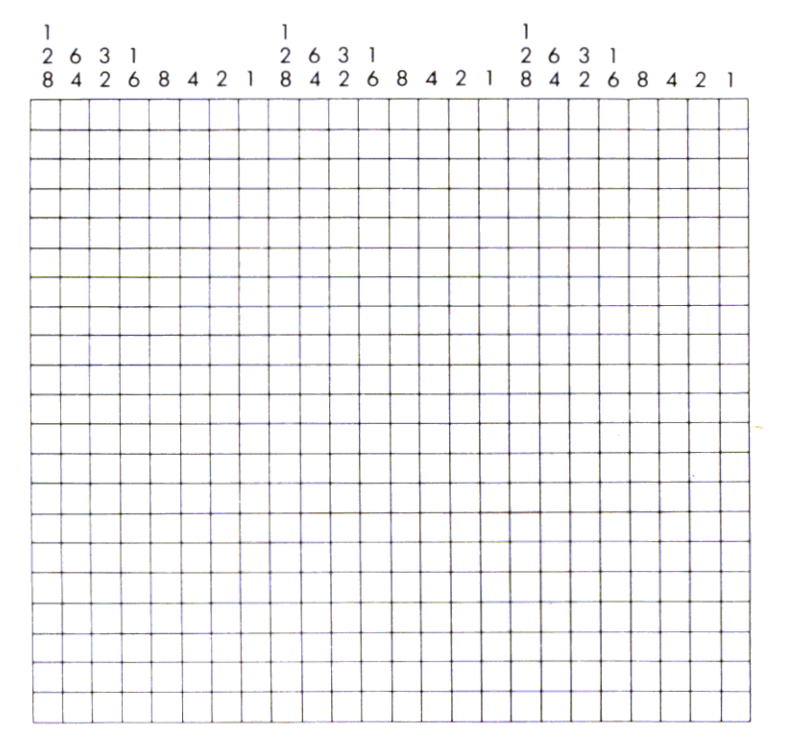


Figure 3-6. Grille de dessin de caractère graphique programmable.

Chaque petit carré correspond à un élément d'image du caractère graphique programmable. Nous avons 24 éléments d'image dans le sens horizontal par 21 dans le sens vertical, soit 504 éléments pour le caractère graphique programmable complet. Pour dessiner un caractère graphique programmable quelconque, on doit colorier certains des éléments d'image avec un programme spécial . . . comment peut-on arriver à commander plus de 500 éléments d'image séparés? À ce point, la programmation est très utile. Au lieu de taper 504 numéros séparés, il suffit de taper seulement 63 numéros pour chaque caractère graphique programmable. Voici comment . . .

CRÉATION DÉTAILLÉE D'UN CARACTÈRE GRAPHIQUE PROGRAMMABLE

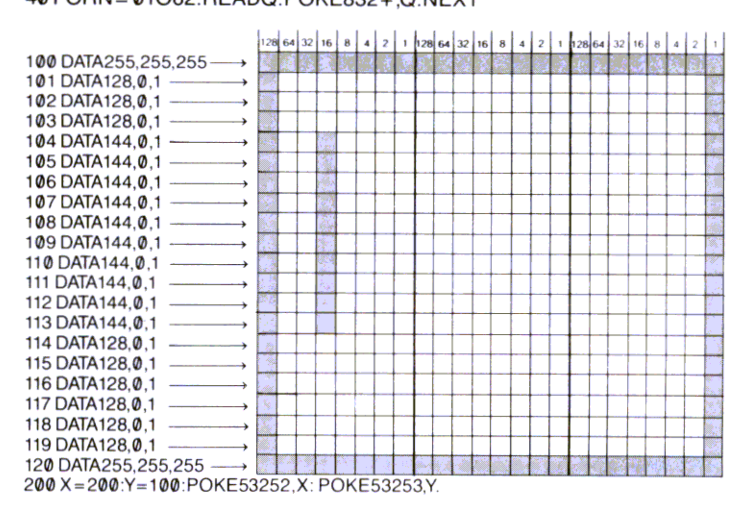
Pour faciliter les choses au maximum, nous avons préparé ce guide détaillé de dessin de caractère graphique programmable.

ÉTAPE 1:

Noter le programme de caractère graphique programmable ci-dessous sur une feuille de papier. Remarquer que la ligne 100 commence une section DATA spéciale du programme qui contient les 63 nombres nécessaires à la création du caractère graphique programmable.

SHIFT CLR/HOME

```
10 PRINT "♥":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+,Q:NEXT
```



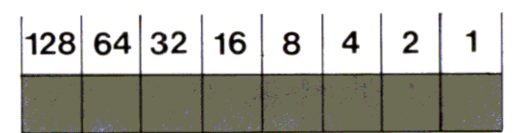
```
100 DATA255,255,255
101 DATA128,0,1
102 DATA128,0,1
103 DATA128,0,1
104 DATA144,0,1
105 DATA144,0,1
106 DATA144,0,1
107 DATA144,0,1
108 DATA144,0,1
109 DATA144,0,1
110 DATA144,0,1
111 DATA144,0,1
112 DATA144,0,1
113 DATA144,0,1
114 DATA128,0,1
115 DATA128,0,1
116 DATA128,0,1
117 DATA128,0,1
118 DATA128,0,1
119 DATA128,0,1
120 DATA255,255,255
200 X=200:Y=100:POKE53252,X: POKE53253,Y
```

ÉTAPE 2:

Colorier les éléments d'image de la grille de caractère graphique programmable de la page 162 (on peut aussi utiliser une feuille de papier graphique, sans oublier qu'un caractère graphique programmable a 24 cases horizontales et 21 verticales). Nous recommandons de dessiner au crayon, sans appuyer, pour pouvoir réutiliser la grille. On peut créer une image quelconque, mais dans cet exemple, nous dessinons une simple case.

ÉTAPE 3:

Jetons un coup d'oeil aux huit premiers éléments d'image. Chaque colonne d'élément d'image porte un numéro (128, 64, 32, 16, 8, 4, 2, 1). L'addition spéciale que nous allons exécuter correspond à un type d'arithmétique binaire utilisé dans la plupart des ordinateurs. Voici une illustration détaillée des 8 premiers éléments d'image, dans le coin supérieur gauche du caractère graphique programmable:



ÉTAPE 4:

Ajouter les nombres des éléments d'image pleins. Ce premier groupe de huit éléments d'image étant complètement rempli, le total est 255.

ÉTAPE 5:

Entrer ce nombre comme première instruction DATA à la ligne 100 du programme de caractère graphique programmable ci-dessous. Entrer 255 pour les deuxième et troisième groupes de huit.

ÉTAPE 6:

Jetons un coup d'oeil aux huit premiers éléments d'image de la deuxième rangée du caractère graphique programmable. Ajoutons les valeurs des éléments d'image pleins. Un seul de ces éléments étant rempli, la valeur totale est de 128. Entrer cette valeur comme premier nombre DATA à la ligne 101.

128	64	32	16	8	4	2	1

ÉTAPE 7:

Ajouter les valeurs du groupe suivant de huit éléments d'image (total de 0 car les éléments sont tous vides) et l'entrer dans la ligne 101. Passer ensuite au groupe suivant d'éléments d'image et répéter le processus pour chaque groupe de 8 éléments (il y a 3 groupes d'éléments sur chacune des 21 rangées). On arrive à un total de 63 nombres. Chaque nombre représente UN groupe de 8 éléments d'image; 63 groupes de huit donnent un total de 504 éléments d'image séparés. Il est peut-être préférable de considérer le programme de la façon suivante . . . chaque ligne du programme représente une rangée du caractère graphique programmable. Chacun des 3 nombres de chaque rangée correspond à un groupe de huit éléments d'image. Chaque nombre indique à l'ordinateur les éléments d'image coloriés et les éléments d'image vierges.

ÉTAPE 8:

COMPRIMER LE PROGRAMME DANS UN ESPACE PLUS PETIT EN EXÉCUTANT ENSEMBLE TOUTES LES INSTRUCTIONS DATA COMME LE MONTRE L'EXEMPLE DE PROGRAMME CI-DESSOUS. Pour une très bonne raison, nous avions demandé précédemment de noter le programme de caractère graphique programmable sur une feuille de papier. Les lignes d'instructions DATA 100 à 120 du programme de l'étape 1 ne servent qu'à indiquer les nombres qui se rapportent au groupe d'élément d'image du caractère graphique programmable. Le programme final peut se "comprimer" ainsi:

```
        SHIFT CLR/HOME
10 PRINT"♥":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT
100 DATA255,255,255,128,0,1,128,0,1,128,0,1, 144,0,1,144,0,1,144,0,1, 144, 0,1
101 DATA144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,128,0, 1,128,0,1
102 DATA128,0,1,128,0,1,128,0,1,128,0,1,255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```

DÉPLACEMENT DU CARACTÈRE GRAPHIQUE PROGRAMMABLE SUR L'ÉCRAN

Après avoir créé un caractère graphique programmable, on peut lui faire faire des évolutions intéressantes. Pour déplacer uniformément le caractère graphique programmable sur l'écran, ajouter les deux lignes suivantes au programme:

```
50 POKE V+5,100:FOR X=24TO255:POKE V+4,X:NEXT:POKE
   V+16,4
55 FOR X=0TO65:POKE V+4,X:NEXT X:POKE V+16,0:GOTO 50
```

LA LIGNE 50 écrit (POKE) la position Y à 100 (pour varier, essayer 50 ou 229). Elle établit ensuite une boucle FOR . . . NEXT qui écrit (POKE) le caractère graphique programmable de la position X 0 à la position X 255. En arrivant à la 255e position, elle écrit (POKE) la position X de droite (POKE V+16,4) nécessaire pour passer au côté droit de l'écran.

LA LIGNE 55 contient une boucle FOR . . . NEXT qui continue à écrire (POKE) le caractère graphique programmable dans les 65 dernières positions sur l'écran. Noter que la valeur X a été remise à zéro, mais comme on a utilisé la valeur X de droite (POKE V+16,2), X recommence sur le côté droit de l'écran.

Cette ligne revient sans cesse sur elle-même (GOTO 50). Si l'on veut que le caractère graphique programmable se déplace une fois sur l'écran et disparaisse, enlever alors GOTO50.

Les lignes suivantes déplacent le caractère graphique programmable dans un sens et dans l'autre:

```
50 POKE V+5,100:FOR X=24TO255:POKE V+4,X:NEXT: POKE
V+16,4:FOR X=0TO65: POKE V+4,X: NEXT X
55 FOR X=65TO0 STEP-1:POKE V+4,X:NEXT:POKE V+16,0: FOR
X=255TO24 STEP-1: POKE V+4,X:NEXT
60 GOTO 50
```

A-t-on compris le déroulement de ces programmes? Celui-ci est identique au précédent, mais quand il arrive à la fin du côté droit de l'écran, il s'inverse lui-même et repart dans l'autre sens à cause de STEP-1 qui indique au programme d'écrire (POKE) le caractère graphique programmable dans les valeurs X de 65 à 0 sur le côté droit de l'écran puis de 255 à 0 sur le côté gauche, à raison de −1 position à la fois.

DÉFILEMENT VERTICAL

Le défilement est un type de déplacement du caractère graphique programmable. Pour faire défiler le caractère dans un sens et dans l'autre dans le sens Y, on doit utiliser une seule ligne. Effacer les lignes 50 et 55 en tapant les numéros de ligne et en appuyant sur la touche **RETURN** , de la façon suivante:

50 (**RETURN**)
55 (**RETURN**)

Entrer de nouveau la ligne 50 de la façon suivante:

```
50 POKE V+4,24:FOR Y=0TO255:POKE V+5,Y:NEXT
```

LA SOURIS DANSANTE— EXEMPLE DE PROGRAMME DE CARACTÈRES GRAPHIQUES PROGRAMMABLES

Il arrive que les techniques décrites dans un manuel de référence de programmeur soient difficiles à comprendre. Pour y remédier, nous avons préparé un programme avec caractères graphiques programmables intitulé "La souris dansante". Ce programme utilise trois caractères graphiques programmables différents pour donner une animation amusante avec des effets sonores. Pour mieux comprendre son déroulement, nous donnons une explication de chaque commande pour assimiler parfaitement la construction du programme:

```
5S = 54272:POKES + 24,15:POKES,220:POKES + 1,68:POKES + 5,15:POKES + 6,215
10 POKES + 7,120:POKES + 8,100:POKES + 12,15:POKES + 13,215
              SHIFT CLR/HOME
15 PRINT" ♥ ":V = 53248:POKEV + 21,1
20 FORS1 = 12288TO12350:READQ1:POKES1,Q1:NEXT
25 FORS2 = 12352TO12414:READQ2:POKES2,Q2:NEXT
30 FORS3 = 12416TO12478:READQ3:POKES3,Q3:NEXT
35 POKEV + 39,15:POKEV + 1,68
                       CTRL 2                        C= 7
40 PRINTTAB(160)" ■ VOICI LA SOURIS DANSANTE! ■ "
45 P = 192
50 FORX = 0TO347STEP3
55 RX = INT(X/256):LX = X - RX*256
60 POKEV,LX:POKEV + 16,RX
70 IFP = 192THENGOSUB200
75 IFP = 193THENGOSUB300
80 POKE2040,P:FORT = 1TO60:NEXT
85 P = P + 1:IFP > 194THENP = 192
90 NEXT
95 END
100 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189, 254,127,255,254
101 DATA63,255,252,31,187,248,3,187,192,1,255,128,3,189,192, 1,231,128,1,255,0
102 DATA31,255,0,0,124,0,0,254,0,1,199,32,3,131,224,7,1,192, 1,192,0,3,192,0
103 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189, 254,127,255,254
104 DATA63,255,252,31,221,248,3,221,192,1,255,128,3,255,192, 1,195,128,1,231,3
105 DATA31,255,255,0,124,0,0,254,0,1,199,0,7,1,128,7,0,204, 1,128, 124,7,128,56
106 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189, 254,127,255,254
107 DATA63,255,252,31,221,248,3,221,192,1,255,134,3, 189,204,1,199,152,1,255,48
108 DATA1,255,224,1,252,0,3,254,0
```

```
109 DATA7,14,Ø,2Ø4,14,Ø,248,56,Ø,112,112,Ø,Ø,6Ø,Ø,-1
2ØØ POKES+4,129:POKES+4,128:RETURN
3ØØ POKES+11,129:POKES+11,128:RETURN
```

LIGNE 5:

S=54272	Fixe la variable S égale à 54272 qui correspond à la position de début de mémoire de la microplaquette de son. À partir de ce point, au lieu d'écrire dans une position de mémoire directe, nous donnerons une instruction POKE S suivie d'une valeur.
POKES+24,15	Identique à POKE 54296,15 qui fixe le volume au niveau le plus élevé.
POKES,220	Identique à POKE 54272,220 qui fixe la basse fréquence de la voix 1 pour une note correspondant approximativement à do dans l'octave 6.
POKES + 1,68	Identique à POKE 54273,68 qui fixe la fréquence élevée dans la voix 1 à une note correspondant approximativement à do dans l'octave 6.
POKES+5,15	Identifque à POKE 54277,15 qui fixe l'attaque/décroissance de la voix 1, dans le cas présent, on utilise le niveau de décroissance maximal, sans attaque, pour obtenir l'effet d'écho.
POKES+6,215	Identique à POKE 54278,215 qui fixe la stabilisation/extinction pour la voix 1 (215 correspond à une combinaison de valeurs de stabilisation et d'extinction).

LIGNE 10:

POKES+7,120	Identique à POKE 54279,120 qui fixe la basse fréquence de la voix 2.
POKES+8,100	Identique à POKE 54280,100 qui fixe la haute fréquence de la voix 2.
POKES+12,15	Identique à POKE 54284,15 qui fixe l'attaque/décroissance de la voix 2 au même niveau que celui de la voix 1 ci-dessus.
POKES+13,215	Identique à POKE 54285,215 qui fixe la stabilisation/extinction de la voix 2 au même niveau que celui de la voix 1 ci-dessus.

LIGNE 15:

PRINT" SHIFT CLR/HOME "	Efface l'écran quand le programe commence.
V = 53248	Définit la variable "V" comme position de départ de la micro-plaquette VIC qui commande les caractères graphiques programmables. Dorénavant, nous définissons les positions de caractère graphique programmable avec V suivi d'une valeur.
POKEV + 21,1	Met en fonction (valide) le caractère graphique programmable 1.

LIGNE 20:

FORS1 = 12288 TO 12350	Dans cette animation, nous allons utiliser un caractère graphique programmable (caractère 0) avec trois jeux de données de caractère graphique pour définir trois formes séparées. Pour arriver à notre animation, nous faisons passer les pointeurs du caractère 0 aux trois points de mémoire oú nous avons stocké les données qui définissent les 3 formes différentes. Le même caractère graphique programmable est rapidement redéfini sans cesse sous 3 formes différentes pour donner l'impression de la souris dansante. On peut définir des douzaines de formes de caractères graphiques programmables dans les instructions DATA et faire passer ces formes dans un ou plusieurs caractères graphiques programmables. On voit donc qu'il n'est pas utile de limiter un caractère graphique à une forme ou vice versa. Un caractère graphique peut avoir plusieurs formes différentes; il suffit de changer le réglage du pointeur de ce caractère à différents points de mémoire oú sont stockées les données du caractère pour les différentes formes. Dans cette ligne, nous avons placé les données de la forme 1 de caractère graphique aux positions de mémoire 12288 à 12350.
READ Q1	Lit 63 nombres dans l'ordre dans les instructions DATA qui commencent à la ligne 100. Q1 est un nom de variable arbitraire qui pourrait aussi être A, Z1 ou une autre variable numérique.

POKES1,Q1	Écrit (POKE) le premier nombre des instructions DATA (le premier "Q1" est 30) dans la première position de mémoire (12288). Cette instruction est identique à POKE12288,30.
NEXT	Indique à l'ordinateur de chercher entre les parties FOR et NEXT de la boucle et d'exécuter les commandes qui s'y trouvent (READQ1 et POKES1,Q1 en utilisant les nombres suivants (NEXT) dans l'ordre). L'instruction NEXT fait donc lire (READ) à l'ordinateur le Q1 suivant (NEXT Q1) des instructions DATA, qui est égal à 0, tout en augmentant S1 de 1 à la valeur suivante qui est 12289. On obtient POKE12289,0 . . . la commande NEXT fait continuer la boucle jusqu'aux dernières valeurs de la série qui sont POKE12350,0.

LIGNE 25:

FORS2 = 12352 TO 12414	La deuxième forme du caractère graphique programmable zéro est définie par l'instruction DATA aux positions 12352 à 12414. NOTER que l'on saute la position 12351 qui correspond à la 64^{e}position utilisée dans la définition du premier groupe de caractères graphiques programmables mais ne contient aucun des nombres de données de caractères graphiques. Quand on définit des caractères graphiques programmables dans des positions consécutives, il faut simplement se rappeler que l'on utilise 64 positions et écrire (POKE) les données dans les 63 premières positions seulement.
READQ2	Lit les 63 nombres qui suivent les nombres utilisés pour la première forme de caractère graphique programmable. Cette instruction READ cherche le nombre suivant dans la zone DATA et commence la lecture des 63 nombres, les uns après les autres.
POKES2,Q2	Écrit (POKE) les données (Q2) dans les positions de mémoire (S2) pour la deuxième forme de caractère graphique programmable qui commence à la position 12352.
NEXT	Remplit la même fonction que la ligne 20 ci-dessus.

LIGNE 30:

FORS2=12416	La troisième forme du caractère zéro est définie par
TO 12478	les instructions DATA aux positions 12416 à 12478.
READQ3	Lit les 63 derniers nombres dans l'ordre dans Q3.
POKES3,Q3	Écrit (POKE) ces nombres dans les positions 12416 à 12478.
NEXT	Fonction identique à celle des lignes 20 et 25.

LIGNE 35:

POKEV+39,15	Fixe la couleur du caractère zéro au gris clair.
POKEV+1,68	Fixe le coin supérieur droit de la case de caractère graphique programmable à la position verticale (Y) 68. À titre de comparaison, la position 50 correspond à la position Y du coin supérieur gauche de l'écran de visualisation.

LIGNE 40:

PRINTTAB(160)	Fait la tabulation de 160 espaces à partir de l'espace de caractère supérieur gauche sur l'écran qui correspond à 4 rangées sous la commande d'effacement . . . on commence ainsi le message d'impression (PRINT) à la 6e ligne de l'écran.
" CTRL WHT	Appuyer simultanément sur la touche CTRL et sur la touche WHT . Si l'on procède à cette opération entre des guillemets, un "E inverse" apparaît. On fixe ainsi au blanc la couleur des éléments qui sont maintenant imprimés.
VOICI LA SOURIS DANSANTE!	Instruction PRINT simple.
C= 7 "	Ramène la couleur au bleu clair à la fin de l'instruction PRINT. Si l'on appuie en même temps sur C= et 7 entre des guillemets, on fait apparaître un losange inverse.

LIGNE 45

P=192	Fixe la variable P à 192. Le nombre 192 est le pointeur à utiliser pour "pointer" le caractère graphique programmable zéro aux positions de mémoire qui commencent à 12288. C'est dans le changement de ce pointeur aux positions des deux autres formes de caractère graphique programmable que réside le secret de l'utilisation d'un de ces caractères pour créer une animation qui se compose de trois formes différentes.

LIGNE 50

FORX=0TO347 STEP3	Fait progresser le déplacement du caractère graphique programmable 3 de X positions à la fois (pour obtenir un déplacement rapide), de la position 0 à la position 347.

LIGNE 55:

RX=INT(X/256)	RX est la partie entière de X/256; RX est arrondi à 0 si X est inférieur à 256; RX devient égal à 1 quand X arrive à la position 256. Nous allons utiliser RX dans un instant pour écrire POKE V+16 avec 0 ou 1 et mettre le côté droit de l'écran en fonction.
LX=X−RX*256	Quand le caractère graphique programmable est à la position X 0, la formule devient: LX = 0 − (0 fois 256), soit 0. Quand le caractère est à la position X 1, la formule est: LX = 1 − (0 fois 256), soit 1. Quand le caractère est à la position X 256, la formule devient: LX = 256 − (1 fois 256), soit 0 qui ramène X à 0; cette opération doit être faite quand on recommence du côté droit de l'écran (POKEV+16,1).

LIGNE 60:

POKEV,LX	On ècrit POKE V par elle-même avec une valeur pour fixer la position X horizontale du caractère programmable 0 sur l'écran. (Voir le tableau de caractères graphiques programmables à la page 176.) Comme on l'indique ci-dessus, la valeur de LX, qui est la position horizontale du caractère, passe de 0 à 255; quand elle arrive à 255, elle revient automatiquement à 0, à cause de l'équation LX à la ligne 55.
POKEV+16,RX	POKEV+16 passe toujours du côté droit de l'écran, au-delà de la position 256, et remet les coordonnées de positionnement horizontal à zéro. RX correspond à 0 ou 1, suivant la position du caractère graphique programmable déterminée par la formule RX à la ligne 55.

LIGNE 70:

IFP=192THEN GOSUB200	Si le pointeur de caractère graphique programmable est fixé à 192 (première forme de caractère), la commande de la forme d'onde du premier effet sonore est fixée à 129 et à 128, suivant la ligne 200.

LIGNE 75:

IFP=193THEN GOSUB300	Si le pointeur de caractère graphique programmable est fixé à 193 (deuxième forme de caractère), la commande de forme d'onde du deuxième effet sonore (voix 2) est fixée à 129 et 128, suivant la ligne 300.

LIGNE 80:

POKE2040,P	Fixe le pointeur de caractère graphique programmable à la position 192 (on a vu que P=192 à la ligne 45. Nous utilisons maintenant la valeur P.)
FORT=1TO60: NEXT	Simple boucle de retard qui détermine le rythme de la souris dansante. (Essayez un rythme plus rapide ou plus lent en augmentant ou diminuant le nombre 60.)

LIGNE 85:

P = P + 1	Nous augmentons maintenant la valeur du pointeur en ajoutant 1 à la valeur initiale de P.
IFP > 194THEN P = 192	Nous désirons seulement pointer le caractère graphique programmable à 3 positions de mémoire: 192 pointe aux positions 12288 à 12350, 193 pointe aux positions 12352 à 12414 et 194 pointe aux positions 12416 à 12478. Cette ligne indique à l'ordinateur de ramener P à 192 dès qu'il devient 195; P ne devient donc jamais vraiment 195. P vaut 192, 193, 194 puis revient à 192; le pointeur indique consécutivement les trois formes de caractère graphique programmable dans les trois groupes de 64 octets des positions de mémoire qui contiennent les instructions DATA.

LIGNE 90:

NEXTX	Quand le caractère graphique programmable a pris l'une des 3 formes différentes définies par les instructions DATA, il peut alors se déplacer sur l'écran. Il saute 3 positions X à la fois (au lieu de défiler uniformément d'une position à la fois, chose également possible). En sautant (STEP) 3 positions à la fois, la souris paraît "danser" plus rapidement sur l'écran. NEXT X correspond à la boucle de position X FOR . . . à la ligne 50.

LIGNE 95

END	Termine le programme quand le caractère graphique programmable sort de l'écran.

LIGNES 100—109

DATA	Les formes du caractère graphique programmable sont lues dans l'ordre dans les nombres des instructions DATA. Les 63 nombres qui constituent la forme 1 de caractère sont d'abord lus, suivis des 63 nombres de la forme 2 puis des nombres de la forme 3. Ces données sont lues en permanence dans les 3 positions de mémoire; après avoir été lues, il suffit au programme de pointer le caractère graphique programmable 0 aux 3 positions de mémoire pour qu'il prenne automatiquement la forme des données de ces positions. Nous pointons le caractère graphique programmable à chacune de 3 positions pour donner l'effet d'animation. Si l'on désire voir l'effet de ces nombres sur chaque caractère graphique programmable, essayer de changer les 3 premiers nombres de la ligne 100 pour avoir 255, 255 et 255. Pour plus de détails, consulter la section sur la définition des formes de caractères graphiques programmables.

LIGNE 200:

POKES+4,129	La commande de forme d'onde fixée à 129 met l'effet sonore en fonction.
POKES+4,128	La commande de forme d'onde fixée à 128 coupe l'effet sonore.
RETURN	Renvoie le programme à la fin de la ligne 70 après le changement des réglages de forme d'onde, pour reprendre le programme.

LIGNE 300:

POKES+11,129	La commande de forme d'onde fixée à 129 met l'effet sonore en fonction.
POKES+11,128	La commande de forme d'onde fixée à 128 coupe l'effet sonore.
RETURN	Renvoie le programme à la fin de la ligne 75 pour reprendre.

TABLEAU DE CRÉATION DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

	CARACTÈRE 0	CARACTÈRE 1	CARACTÈRE 2	CARACTÈRE 3	CARACTÈRE 4	CARACTÈRE 5	CARACTÈRE 6	CARACTÈRE 7
Mise en fonction caractère	V+21,1	V+21,2	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128
Mise en mémoire (réglage pointeurs)	2040, 192	2041, 193	2042, 194	2043, 195	2044, 196	2045, 197	2046, 198	2047, 199
Positions pour élément d'image (12288—12798)	12288 à 12350	12352 à 12414	12416 à 12478	12480 à 12542	12544 à 12606	12608 à 12670	12672 à 12734	12736 à 12798
Couleur du caractère	V+39,C	V+40,C	V+41,C	V+42,C	V+43,C	V+44,C	V+45,C	V+46,C
Réglage position X gauche (0—255)	V+0,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
Réglage position X droite (0—255)	V+16,1 V+0,X	V+16,2 V+2,X	V+16,4 V+4,X	V+16,8 V+6,X	V+16,16 V+8,X	V+16,32 V+10,X	V+16,64 V+12,X	V+16,128 V+14,X
Réglage position Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
Étalement horizontal du caractère/X	V+29,1	V+29,2	V+29,4	V+29,8	V+29,16	V+29,32	V+29,64	V+29,128
Étalement vertical du caractère/Y	V+23,1	V+23,2	V+23,4	V+23,8	V+23,16	V+23,32	V+23,64	V+23,128
Mise en fonction mode multicolore	V+28,1	V+28,2	V+28,4	V+28,8	V+28,16	V+28,32	V+28,64	V+28,128
Multicolore 1 (première couleur)	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C
Multicolore 2 (deuxième couleur)	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C
Réglage priorité des caractères	Les caractères graphiques programmables de numéro plus bas ont toujours la priorité d'affichage sur les caractères de numéro plus élevé. Par exemple, le caractère 0 est prioritaire sur tous les autres caractères; le caractère 7 a la priorité la plus basse. Les caractères de numéro plus bas paraissent donc toujours se déplacer devant ou au-dessus des caractères de numéro plus élevé.							
Collision (entre caractères)	V+30	IF PEEK(V+30)ANDX=X THEN [action]						
Collision (caractère avec arrière-plan)	V+31	IF PEEK(V+31)ANDX=X THEN [action]						

REMARQUES SUR LA CRÉATION DES CARACTÈRES GRAPHIQUES PROGRAMMABLES

Autres pointeurs de mémoire de caractères et positions de mémoire avec tampon de cassette

Mise en mémoire (réglage des pointeurs)	CARAC-TÈRE 0 2040,13	CARAC-TÈRE 1 2041,14	CARAC-TÈRE 2 2042,15	Avec 1 à 3 caractères graphiques programmables, on peut utiliser ces positions de mémoire dans le tampon de cassette (832 à 1023). Avec plus de 3 caractères graphiques programmables, nous recommandons d'utiliser les positions de 12288 à 12798 (voir tableau).
Position des éléments d'image de caractère pour blocs 13-15	832 à 894	896 à 958	960 à 1022	

MISE EN FONCTION DES CARACTÈRES GRAPHIQUES PROGRAMMABLES:

On peut mettre en fonction n'importe quel caractère graphique programmable avec POKE V+21 et le nombre du tableau . . . mais la mise en fonction d'un seul caractère coupe les autres. Pour mettre en fonction deux ou plusieurs caractères, ajouter les nombres des caractères correspondants (exemple: POKEV+21,6 met en fonction les caractères graphiques programmables 1 et 2). La méthode suivante permet de mettre un caractère en et hors fonction sans affecter les autres (pratique en animation).

EXEMPLE:

Pour couper seulement le caractère 0, taper: POKE V+21,PEEK V+21AND(255−1). Remplacer le nombre 1 dans (255−1) par 1,2,4,8,16,32,64 ou 128 (pour les caractères graphiques programmables 0-7). Pour revalider le caractère et ne pas affecter les autres présentement en fonction, taper POKE V+21, PEEK (V+21)OR 1 et remplacer OR 1 par OR2 (caractère 2), OR 4 (caractère 3), etc.

VALEURS DES POSITIONS X APRÈS 255:

Les positions X vont de 0 à 255; elles recommencent ensuite de 0 à 255. Pour placer un caractère graphique programmable au-delà de la position X 255 à l'extrême droite de l'écran, on doit d'abord taper POKE V+16, tel qu'il est indiqué, puis POKE avec une nouvelle valeur X de 0 à 63 afin de placer le caractère dans l'une des positions X de l'extrême droite de l'écran. Pour revenir aux positions 0-255, taper POKE V+16,0 et POKE avec une valeur X de 0 à 255.

VALEURS DES POSITIONS DE Y:

Les positions Y vont de 0 à 255, y compris 0 à 49 en haut et en dehors de la zone visible, de 50 à 229 dans la zone visible et de 230 à 255 en bas et en dehors de cette zone.

COULEURS DES CARACTÈRES GRAPHIQUES PROGRAMMABLES:

Pour rendre le caractère graphique programmable 0 blanc, taper POKE V+39,1 (utiliser le réglage POKE couleur indiqué dans le tableau et les codes de couleur individuels ci-dessous):

0—NOIR	4—VIOLET	8—ORANGE	12—GRIS MOYEN
1—BLANC	5—VERT	9—BRUN	13—VERT CLAIR
2—ROUGE	6—BLEU	10—ROUGE CLAIR	14—BLEU CLAIR
3—TURQUOISE	7—JAUNE	11—GRIS FONCÉ	15—GRIS CLAIR

POSITION DE MÉMOIRE:

On doit "réserver" un bloc séparé de 64 octets de nombres dans la mémoire de l'ordinateur pour chaque caractère graphique programmable; 63 octets servent aux données du caractère. Les réglages de mémoire indiqués ci-dessous sont recommandés pour les réglages de pointeurs de caractère du tableau précédent. L'utilisateur doit définir chaque caractère graphique programmable qui est unique en son genre. Pour que tous les caractères soient identiques, les pointer vers le même registre de caractère.

RÉGLAGES DIFFÉRENTS DE POINTEURS DE CARACTÈRE:

Ces réglages de pointeurs de caractère graphique programmable ne sont que des recommandations.

Attention: on peut fixer les pointeurs de caractère en tout point de la mémoire vive RAM, mais si on les règle trop bas dans la mémoire, un long programme BASIC peut recouvrir les données de caractères ou vice versa. Pour protéger un programme BASIC particulièrement long de "l'écrasement" par les données de caractères, on peut fixer les caractères graphiques programmables dans une zone plus élevée de la mémoire (par exemple, 2040,192 pour le caractère 0 aux positions 12288 à 12350; 2041,193 aux positions 12352 à 12414 pour le caractère 1, etc. En ajustant les positions de mémoire d'où les caractères graphiques programmables tirent leurs données, on peut définir jusqu'à 64 caractères différents et un programme BASIC de taille respectable. Dans ce but, définir plusieurs formes de caractères graphiques programmables dans les instructions données puis redéfinir un caractère particulier en changeant le pointeur pour que le caractère

utilisé soit "pointé" à différentes zones de mémoire contenant des données d'image différentes. Voir "La souris dansante" pour comprendre ce processus. Si l'on désire que 2 ou plusieurs caractères graphiques programmables aient la même forme (on peut encore changer la position et la couleur de chaque caractère), utiliser un pointeur de caractère et une position de mémoire identiques pour les caractères correspondants (par exemple, on peut pointer les caractères 0 et 1 à la même position avec POKE 2040,192 et POKE 2041,192).

PRIORITÉ:

La priorité fait se déplacer un caractère graphique programmable devant ou derrière un autre sur l'écran. Les caractères à priorité plus élevée semblent se déplacer devant ou par-dessus les caractères à priorité plus basse. Les caractères à numéro plus bas sont prioritaires sur les caractères à numéro plus élevé. Le caractère 0 a la priorité sur tous les autres. Le caractère 7 a la priorité la plus basse. Le caractère 1 a la priorité sur les caractères 2-7, etc. Si l'on place deux caractères graphiques programmables à la même position, le caractère à priorité plus élevée apparaît devant l'autre. Le caractère à priorité plus basse peut être masqué ou être vu par transparence (derrière) le caractère à priorité plus élevée.

UTILISATION DU MODE MULTICOLORE:

On peut créer des caractères graphiques programmables multicolores, mais l'utilisation du mode multicolore exige que l'on utilise des paires d'éléments d'image à la place d'éléments individuels dans l'image du caractère (chaque point ou bloc de couleur du caractère graphique programmable se compose alors de 2 éléments d'image juxtaposés). On dispose d'un choix de 4 couleurs: couleur du caractère graphique programmable (tableau ci-dessus), multicolore 1, multicolore 2 et couleur d'arrière-plan (on obtient l'arrière-plan avec des réglages "0" qui permettent de voir la couleur d'arrière-plan par transparence). Considérons un bloc horizontal de 8 éléments dans une image de caractère graphique programmable. La couleur de chaque paire d'éléments d'image est déterminée de la façon suivante, suivant que les éléments de gauche et(ou) de droite sont coloriés:

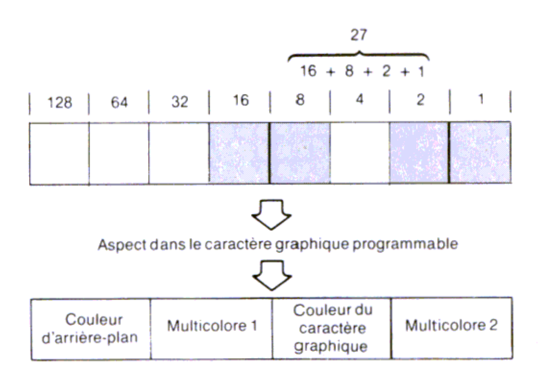
ARRIÈRE-PLAN (Les deux éléments d'image sont vierges (zéro) et laissent voir la couleur intérieure de l'écran (arrière-plan) par transparence.)

MULTICOLORE 1 (Le coloriage de l'élément d'image de droite dans une paire d'éléments les fixe tous deux à multicolore 1.)

COULEUR DE CARACTÈRE GRAPHIQUE (Le coloriage de l'élément d'image de gauche dans une paire d'éléments les fixe tous deux à la couleur du caractère graphique programmable.)

MULTICOLORE 2 (Le coloriage des deux éléments dans une paire les fixe tous deux à multicolore 2.)

Jetons un coup d'oeil à la rangée horizontale de 8 éléments d'image ci-dessous. Ce bloc fixe les deux premiers éléments à la couleur d'arrière-plan, les deux suivants à multicolore 1, les deux éléments qui suivent à la couleur du caractère graphique programmable et les deux derniers à multicolore 2. La couleur de chaque paire d'éléments d'image dépend des bits coloriés et vierges dans chaque paire, comme on l'a vu ci-dessus. Après avoir déterminé les couleurs désirées dans chaque paire d'éléments d'image, il faut ensuite ajouter les valeurs des éléments coloriés dans le bloc de 8 éléments et écrire (POKE) le nombre obtenu dans la position de mémoire correspondante. Par exemple, si la rangée de 8 éléments d'image ci-dessous correspond au premier bloc d'un caractère graphique programmable qui commence à la position de mémoire 832, la valeur des éléments d'image coloriés est 16+8+2+1 = 27, on doit donc écrire POKE 832,27.



COLLISION:

On peut détecter la collision d'un caractère graphique programmable avec un autre à l'aide de la ligne suivante: IF PEEK(V+30)ANDX=XTHEN [insérer ici l'action]. Cette ligne vérifie si un caractère graphique programmable donné est entré en collision avec un autre. X vaut respectivement 1, 2, 4, 8, 16, 32, 64 et 128 pour les caractères graphiques programmables 0, 1, 2, 3, 4, 5, 6 et 7. Pour vérifier si le caractère graphique programmable est entré en collision avec un "caractère d'arrière-plan", utiliser la ligne suivante: IF PEEK(V+31)ANDX=XTHEN [insérer l'action ici].

UTILISATION DES CARACTÈRES GRAPHIQUES DANS LES INSTRUCTIONS DATA

Le programme suivant permet de créer un caractère graphique programmable à l'aide d'éléments vierges et de cercles pleins (SHIFT Q) dans les instructions DATA. Le caractère graphique programmable et les nombres écrits (POKE) dans les registres de données de caractère graphique programmable sont affichés.

```
         SHIFT CLR/HOME
10 PRINT"  ":FORI=0TO63:POKE832+I,0:NEXT
20 GOSUB60000
999 END
60000 DATA"         ●●●●●●●        "
60001 DATA"       ●●●●●●●●●●●      "
60002 DATA"      ●●●●●●●●●●●●●     "
60003 DATA"      ●●●●●   ●●●●●     "
60004 DATA"     ●●●●● ●●●  ●●●●    "
60005 DATA"     ●●●●● ●●● ●●●●●    "
60006 DATA"     ●●●●● ●●●  ●●●●    "
60007 DATA"      ●●●●●   ●●●●●     "
60008 DATA"      ●●●●●●●●●●●●●     "
60009 DATA"      ●●●●●●●●●●●●●     "
60010 DATA"      ● ●●●●●●●●● ●     "
60011 DATA"       ● ●●●●●●● ●      "
60012 DATA"       ●  ●●●●●  ●      "
60013 DATA"        ●  ●●●  ●       "
60014 DATA"        ●  ●●●  ●       "
60015 DATA"         ●  ●  ●        "
60016 DATA"         ●  ●  ●        "
60017 DATA"          ●●●●●         "
60018 DATA"          ●●●●●         "
60019 DATA"          ●●●●●         "
60020 DATA"           ●●●          "
60100 V=53248:POKEV,200:POKEV+1,100:POKEV+21,1:POKEV+39, 14:POKE2040,13
60105 POKEV=23,1:POKEV+29,1
60110 FORI=0TO20:READA$:FORK=0TO2:T=0:FORJ=0TO7:B=0
60140 IFMID$(A$,J+K*8+1,1)="●"THENB=1
60150T=T+B*2↑(7-J):NEXT:PRINTT;:POKE832+I*3+K, T:NEXT:PRINT:NEXT
60200 RETURN
```

CHAPITRE 4

PROGRAMMATION DU SON ET DE LA MUSIQUE AVEC LE COMMODORE 64

INTRODUCTION

Cet ordinateur Commodore est équipé de l'un des synthétiseurs de musique électronique parmi les plus perfectionnés à ce jour. Il comprend trois voix totalement adressables, **L'ATTAQUE/DÉCROISSANCE/STABILISATION/EXTINCTION**, le filtrage, la modulation et le bruit blanc. On peut disposer directement de toutes ces possibilités à l'aide de quelques instructions et fonctions en BASIC ou en langage d'assemblage. On peut ainsi créer des sons et des chants très complexes avec des programmes relativement simples à préparer.

La présente section du guide de référence du programmeur a été préparée pour aider à étudier toutes les possibilités de la microplaquette "SID" 6581, qui remplit les fonctions de synthétiseur de son et de musique dans l'ordinateur Commodore. Nous passerons en revue les notions de musique et les aspects pratiques conduisant à la matérialisation de ces notions dans des chants réels sur l'ordinateur Commodore.

Il n'est pas utile d'être un programmeur chevronné ou un grand musicien pour arriver à des résultats intéressants avec ce synthétiseur de musique. La présente section contient de nombreux exemples de programmation avec explications complètes pour mettre l'utilisateur sur la bonne voie.

On accède au générateur de son en inscrivant des instructions (POKE) aux positions spécifiées de mémoire. L'annexe O donne une liste complète des positions utilisées. Nous étudierons chaque concept en détail. À la fin de cette section, on doit être en mesure de créer une variété quasi infinie de sons et de procéder à des expériences sonores personnelles.

Chaque section de ce chapitre commence avec un exemple et une description ligne par ligne de chaque programme qui vise à montrer l'utilisation de la caractéristique étudiée. On peut lire l'explication technique par la suite si l'on veut en savoir davantage sur ce sujet.

L'instruction POKE est l'élément essentiel des programmes de son. Elle fixe la position indiquée de mémoire (MEM) à une valeur spécifique (NUM).

POKE, MEM, NUM.

Les positions de mémoire (MEM) utilisées dans la synthèse de la musique commencent à 54272 ($D400) dans le Commodore 64. Les positions de mémoire 54272 à 54296 comprises correspondent aux positions POKE que l'on doit connaître quand on utilise la topographie de registre de la microplaquette 6581 (SID). Pour utiliser les positions ci-dessus, il suffit de se rappeler simplement la position 54272 et de lui ajouter un nombre de 0 à 24. On peut ainsi écrire (POKE) dans toutes les positions nécessaires de 54272 à 54296 de la microplaquette SID. Les nombres (NUM) utilisés dans l'instruction POKE doivent être compris entre 0 et 255 inclus.

Quand on s'est un peu familiarisé avec la création musicale, on peut progresser davantage en utilisant la fonction PEEK. PEEK est égale à la valeur présentement dans la position indiquée de mémoire.

X=PEEK(MEM)

La valeur de la variable X est fixée égale au contenu présent de la position de mémoire MEM.

Les programmes contiennent évidemment d'autres commandes BASIC. Pour plus de détails à cet égard, se reporter à la section sur les instructions BASIC de ce manuel.

Entrons immédiatement dans le vif du sujet et essayons un programme simple avec une seule des trois voix. L'ordinateur est-il prêt? Taper NEW puis le programme suivant. Le sauvegarder sur disque ou sur cassette avec le DATASETTE™ de Commodore. Ensuite, l'exécuter (RUN).

EXEMPLE DE PROGRAMME 1:

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT:REM EFFACEMENT DE LA MICROPLAQUETTE
    DE SON
20 POKES+5,9:POKES+6,0
30 POKES+24,15                    :REM REGLAGE DU VOLUME AU MAXIMUM
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Nous donnons ci-après une explication ligne par ligne du programme que l'on vient de taper. S'y reporter si l'on veut approfondir des parties du programme que l'on ne comprend pas parfaitement.

EXPLICATION LIGNE PAR LIGNE DE L'EXEMPLE DE PROGRAMME 1:

Ligne(s)	Description
5	Fixe S au début de la microplaquette de son.
10	Efface tous les registres de microplaquette de son.
20	Fixe l'attaque/décroissance pour la voix 1 (A=0,D=9).
	Fixe la stabilisation/extinction pour la voix 1 (S=0,R=0).
30	Fixe le volume au maximum.
40	Lit la haute fréquence, la basse fréquence et la durée de la note.
50	Si la haute fréquence est inférieure à zéro, le chant est terminé.
60	Inscrit la haute fréquence et la basse fréquence de la voix 1.
70	Déclenche la forme d'onde en dents de scie pour la voix 1.
80	Boucle le minutage de la durée de la note.
90	Arrête la forme d'onde en dents de scie pour la voix 1.
100	Revient à la note suivante.
100-180	Données du chant: haute fréquence, basse fréquence, durée (nombre de comptes) pour chaque note.
190	Dernière note du chant et uns négatifs indiquant la fin du chant.

COMMANDE DE VOLUME

Le registre 24 de la microplaquette contient la commande de volume globale. On peut régler le volume entre 0 et 15. Nous expliquerons l'utilité des quatre autres bits par la suite. Pour le moment, il suffit de savoir que le volume se règle de 0 à 15. Jeter un coup d'oeil à la ligne 30 pour comprendre son réglage dans l'exemple de programme 1.

FRÉQUENCES DES ONDES SONORES

Le son est créé par le déplacement ondulatoire de l'air. Si l'on jette une pierre dans une mare, il se crée des ondes au point d'impact. Quand il se forme des ondes analogues dans l'air, nous pouvons les entendre. Si nous mesurons la durée entre la crête d'une onde et la suivante, nous obtenons le nombre de secondes d'un cycle de l'onde (n = nombre de secondes). L'inverse de ce nombre (1/n) donne le nombre de cycles par seconde plus couramment connu sous le nom de "fréquence". Cette fréquence détermine la hauteur du son créé par des ondes sonores.

Dans l'ordinateur Commodore, le générateur de son utilise deux positions pour déterminer la fréquence. L'annexe E donne les valeurs des fréquences nécessaires pour reproduire huit octaves complètes de notes de musique. Pour créer une fréquence en dehors de celles indiquées dans la table des notes, utiliser la fréquence de sortie (F_{sortie}) et la

formule suivante pour déterminer la fréquence (F_n) du son à créer. Ne pas oublier que chaque note doit avoir un nombre de haute fréquence et de basse fréquence.

$$F_n = F_{sortie}/.06097$$

Après avoir déterminé F_n pour la "nouvelle" note, il faut créer les valeurs de haute et de basse fréquence pour cette note. Dans ce but, arrondir d'abord F_n pour éliminer les chiffres à la droite du point décimal. Il nous reste une valeur entière. On peut maintenant établir la position de haute fréquence (F_h) avec la formule $F_h = INT(F_n/256)$ et la position de basse fréquence (F_b) qui est $F_b = F_n - (256*F_h)$.

À ce stade, on a déjà reproduit une voix de l'ordinateur. Si on veut s'arrêter ici, on peut se procurer une copie d'un air de musique et se préparer à diriger un "orchestre informatisé" dans une salle de concert "à domicile".

UTILISATION DE PLUSIEURS VOIX

L'ordinateur Commodore possède trois voix à commandes indépendantes (oscillateurs). Dans notre premier exemple de programme, nous avons utilisé l'une d'elles. Par la suite, nous apprendrons à changer la qualité du son produit par les voix. Pour le moment, mettons les trois voix en fonction.

L'exemple suivant de programme montre la transposition d'une partition pour "l'orchestre informatisé". Taper ce programme puis le sauvegarder (SAVE) sur cassette avec le DATASSETTE™ ou sur disque. Ne pas oublier de taper NEW avant d'introduire le programme au clavier.

EXEMPLE DE PROGRAMME 2:

```
10 S=54272:FORL=STOS+24:POKEL,0:NEXT
20 DIMH(2,200),L(2,200),C(2,200)
30 DIMFQ(11)
40 V(0)=17:V(1)=65:V(2)=33
50 POKES+10,8:POKES+22,128:POKES+23,244
60 FORI=0TO11:READFQ(I):NEXT
100 FORK=0TO2
110 I=0
120 READNM
130 IFNM=0THEN250
140 WA=V(K):WB=WA-1:IFNM<0THENNM=-NM:WA=0:WB=0
150 DR%=NM/128:OC%=(NM-128*DR%)/16
160 NT=NM-128*DR%-16*OC%
170 FR=FQ(NT)
```

```
180 IFOC%=7THEN200
190 FORJ=6TOOC%STEP-1:FR=FR/2:NEXT
200 HF%=FR/256:LF%=FR-256*HF%
210 IFDR%=1THENH(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA: I=I+1:GOTO120
220 FORJ=1TODR%-1:H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA: I=I+1:NEXT
230 H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WB
240 I=I+1:GOTO120
250 IFI>IMTHENIM=I
260 NEXT
500 POKES+5,0:POKES+6,240
510 POKES+12,85:POKES+13,133
520 POKES+19,10:POKES+20,197
530 POKES+24,31
540 FORI=0TOIM
550 POKES,L(0,I):POKES+7,L(1,I):POKES+14,L(2,I)
560 POKES+1,H(0,I):POKES+8,H(1,I):POKES+15,H(2,I)
570 POKES+4,C(0,I):POKES+11,C(1,I):POKES+18,C(2,I)
580 FORT=1TO80:NEXT:NEXT
590 FORT=1TO200:NEXT:POKES+24,0
600 DATA34334,36376,38539,40830
610 DATA43258,45830,48556,51443
620 DATA54502,57743,61176,64814
1000 DATA594,594,594,596,596
1010 DATA1618,587,592,587,585,331,336
1020 DATA1097,583,585,585,585,587,587
1030 DATA1609,585,331,337,594,594,593
1040 DATA1618,594,596,594,592,587
1050 DATA1616,587,585,331,336,841,327
1060 DATA1607
1999 DATA0
2000 DATA583,585,583,583,327,329
2010 DATA1611,583,585,578,578,578
2020 DATA196,198,583,326,578
2030 DATA326,327,329,327,329,326,578,583
2040 DATA1606,582,322,324,582,587
2050 DATA329,327,1606,583
2060 DATA327,329,587,331,329
2070 DATA329,328,1609,578,834
2080 DATA324,322,327,585,1602
2999 DATA0
```

```
3000 DATA567,566,567,304,306,308,310
3010 DATA1591,567,311,310,567
3020 DATA306,304,299,308
3030 DATA304,171,176,306,291,551,306,308
3040 DATA310,308,310,306,295,297,299,304
3050 DATA1586,562,567,310,315,311
3060 DATA308,313,297
3070 DATA1586,567,560,311,309
3080 DATA308,309,306,308
3090 DATA1577,299,295,306,310,311,304
3100 DATA562,546,1575
3999 DATA0
```

Nous donnons ci-dessous une explication ligne par ligne de l'exemple du programme 2. Pour le moment, nous nous intéressons à la commande des trois voix.

EXPLICATION LIGNE PAR LIGNE DE L'EXEMPLE DE PROGRAMME 2:

Ligne(s)	Description
10	Fixe S égal au départ de la microplaquette de son et efface tous les registres de cette microplaquette.
20	Dimensionne les tableaux pour qu'ils contiennent le chant à raison de 1/16 de mesure par position.
30	Dimensionne le tableau pour qu'il contienne la fréquence de base de chaque note.
40	Mémorise l'octet de commande de forme d'onde pour chaque voix.
50	Fixe la largeur d'impulsion haute pour la voix 2. Fixe la haute fréquence pour la coupure du filtre. Fixe la résonnance pour le filtre et pour filtrer la voix 3.
60	Lit la fréquence de base pour chaque note.
100	Commence la boucle de décodage pour chaque voix.
110	Initialise le pointeur sur le tableau d'activités.
120	Lit la note codée.
130	Si la note codée est nulle, passe à la voix suivante.
140	Fixe les commandes de forme d'onde à la voix correspondante. En cas de silence, fixe les commandes de forme d'onde à 0.
150	Décode la durée et l'octave.
160	Décode la note.

Ligne(s)	Description
170	Extrait la fréquence de base pour cette note.
180	Pour l'octave la plus haute, saute à la boucle de division.
190	Divise la fréquence de base par 2 le nombre approprié de fois.
200	Extrait les octets de haute et basse fréquence.
210	Pour une double croche, fixe le tableau d'activités: haute fréquence, basse fréquence et commande de forme d'onde (voix en fonction).
220	Pour toute la note, dernière mesure exceptée, fixe le tableau d'activités: haute fréquence, basse fréquence, commande de forme d'onde (voix en fonction).
230	Pour la dernière mesure de la note, fixe le tableau haute fréquence, basse fréquence, commande de forme d'onde (voix hors fonction).
240	Augmente le pointeur et le remet sur le tableau d'activités. Extrait la note suivante.
250	Si la note est plus longue que précédemment, réétablit le nombre d'activités.
260	Revient à la voix suivante.
500	Fixe l'attaque/décroissance pour la voix 1 (A=0, D=0). Fixe la stabilisation/extinction pour la voix 1 (S=15, R=0).
510	Fixe l'attaque/décroissance pour la voix 2 (A=5, D=5). Fixe la stabilisation/extinction pour la voix 2 (S=8, R=5).
520	Fixe l'attaque/décroissance pour la voix 3 (A=0, D=10). Fixe la stabilisation/extinction pour la voix 3 (S=12, R=5).
530	Fixe le volume à 15 et le filtrage passe-bas.
540	Commence la boucle pour chaque 1/16 de mesure.
550	Inscrit (POKE) la basse fréquence du tableau d'activités pour toutes les voix.
560	Inscrit la haute fréquence du tableau d'activités pour toutes les voix.
570	Inscrit la commande de forme d'onde pour le tableau d'activités de toutes les voix.
580	Boucle le minutage à 1/16 de mesure et retourne au 1/16 de mesure suivant.
590	Pause, puis coupure du volume.
600-620	Données de fréquences de base.
1000-1999	Données de la voix 1.
2000-2999	Données de la voix 2.
3000-3999	Données de la voix 3.

Nous avons déterminé les valeurs des instructions de données à l'aide de la table de notes de l'annexe E et du tableau ci-dessous:

TYPE DE NOTE	DURÉE
Double croche (1/16)	**128**
Croche (1/8)	**256**
Croche pointée (1/8)	**384**
Noire (1/4)	**512**
Noire + double croche (1/4+ 1/16)	**640**
Noire pointée (1/4)	**768**
Blanche (1/2)	**1024**
Blanche + double croche (1/2+ 1/16)	**1152**
Blanche + croche (1/2+ 1/8)	**1280**
Blanche pointée (1/2)	**1536**
Ronde	**2048**

Le nombre de la note pris dans la table de notes s'ajoute à la durée ci-dessus. On peut ensuite entrer chaque note avec un seul nombre qui est décodé par le programme. Cette méthode n'est que l'une de celles qui permettent de coder les valeurs de notes. On peut en mettre une autre au point que l'on trouvera plus acceptable. Pour le codage d'une note, nous utiliserons ici la formule suivante:

1) La durée (le nombre de 1/16 d'une mesure) est multipliée par 8.
2) On ajoute le résultat de l'étape 1 à l'octave choisie (0-7).
3) On multiplie le résultat de l'étape 2 par 16.
4) On ajoute le choix de la note (0-11) au résultat de l'étape 3.

Autrement dit:

((((D*8)+O)*16)+N)

Dans laquelle D = durée, O = octave et N = note.

On obtient un silence en utilisant la valeur négative du nombre de durée (nombre de 1/16 d'une mesure * 128).

CONTRÔLE DE PLUSIEURS VOIX

Quand on s'est familiarisé avec l'utilisation de plusieurs voix, on s'aperçoit que l'on doit coordonner la synchronisation des trois voix. Dans ce programme, on y parvient de la façon suivante:

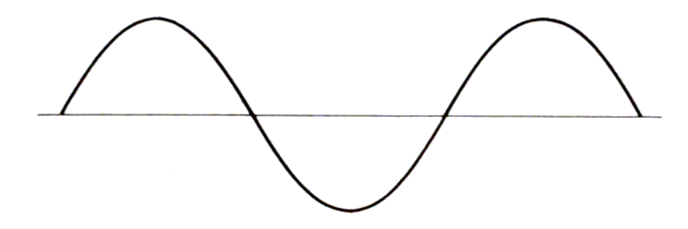
1) On divise chaque mesure en 16 parties.
2) On mémorise les conditions dans chaque intervalle de 1/16 dans trois tableaux séparés.

On calcule les octets de haute et de basse fréquence en divisant les fréquences de l'octave la plus élevée par 2 (lignes 180 et 190). L'octet de commande de forme d'onde est un signal de départ pour commencer une note ou en continuer une autre en cours. C'est aussi un signal d'arrêt d'une note. On procède au choix de la forme d'onde une fois pour chaque voix à la ligne 40.

Cette méthode ne représente qu'une manière de commander plusieurs voix. On peut en mettre soi-même d'autres au point. On doit être maintenant en mesure de lire les notes pour les trois voix sur une partition de musique quelconque.

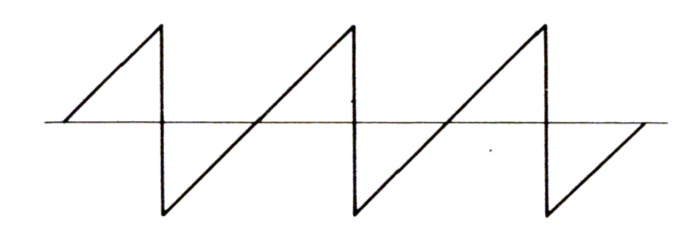
CHANGEMENT DES FORMES D'ONDE

Le timbre correspond à la tonalité d'un son. Il se détermine essentiellement par sa "forme d'onde". Si l'on se rappelle l'exemple de la pierre jetée dans l'eau, on sait que les ondes se répandent uniformément à la surface. Ces ondes ressemblent presque à la première onde sonore dont nous allons parler qui est l'onde sinusoïdale (voir ci-dessous).



Pour un peu plus de clarté, revenons au premier exemple de programme pour étudier différentes formes d'onde. En effet, on peut mieux discerner les changements en n'utilisant qu'une seule voix. Charger (LOAD) le premier programme de musique que nous

avons précédemment tapé à partir du disque ou du DATASETTE™ et l'exécuter (RUN) de nouveau. Ce programme utilise la forme d'onde en dents de scie ci-dessous



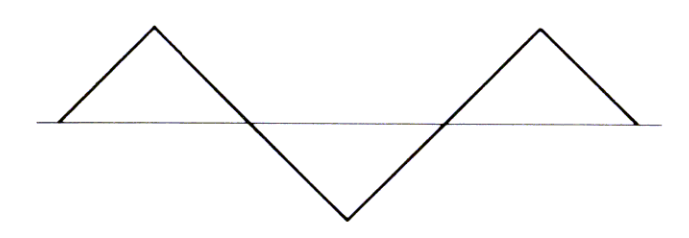
qui vient du dispositif générateur de son de la microplaquette (SID) 6581. À la ligne 70, changer le nombre de départ de la note de 33 à 17 et, à la ligne 90, changer le nombre d'arrêt de la note de 32 à 16. Le programme a maintenant l'aspect suivant:

EXEMPLE DE PROGRAMME 3 (EXEMPLE 1 MODIFIÉ):

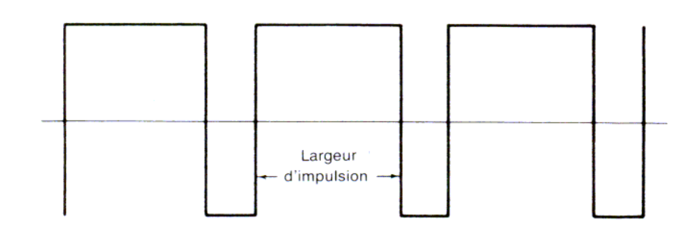
```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,9:POKES+6,0
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Exécuter (RUN) ce programme.

On peut remarquer que la qualité sonore a changée. Elle est moins nasillarde et moins creuse, car nous avons changé la forme d'onde en dents de scie pour avoir une onde triangulaire (voir ci-dessous).



La troisième forme d'onde musicale est l'impulsion variable (voir ci-dessous).



Cette onde est rectangulaire; on détermine la longueur du cycle d'impulsion en définissant la proportion de l'onde à l'état haut. Dans ce but, on utilise les registres 2 et 3 pour la voix 1. Le registre 2 correspond à l'octet bas de la largeur d'impulsion (L_{li} =0 à 255). Le registre 3 correspond aux 4 bits (H_{li} =0 à 15).

Ces registres réunis spécifient un nombre de 12 bits pour la largeur d'impulsion que l'on peut déterminer avec la formule suivante:

$$Limp._n = H_{li}*256+L_{li}$$

L'équation suivante détermine la largeur d'impulsion:

$$Limp._{sortie} = (Limp._n/40.95)\%$$

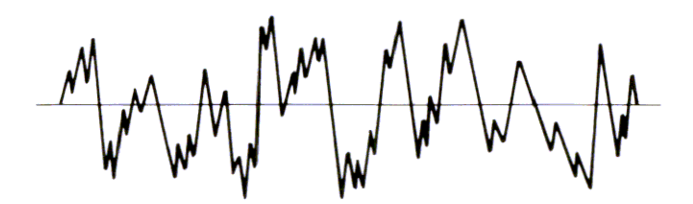
Si Limp.$_n$ vaut 2048, on obtient une onde carrée. Dans ce cas, le registre 2(L_{li}) est égal à 0 et le registre 3 (H_{li}) est égal à 8.

Ajouter maintenant la ligne suivante au programme:

15 POKES+3,8:POKES+2,0

Changer ensuite à 65 le nombre de départ à la ligne 70 et le nombre d'arrêt de la ligne 90 à 64. Exécuter (RUN) le programme. Changer maintenant la largeur d'impulsion haute (registre 3 de la ligne 15) de 8 à 1. On peut certainement remarquer le changement de la qualité sonore!

La dernière forme d'onde obtenue correspond au bruit blanc (voir ci-dessous).

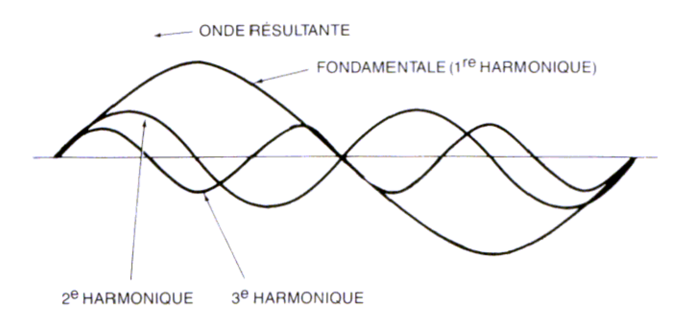


Le bruit blanc s'utilise essentiellement dans les effets sonores. Pour l'entendre, changer à 129 le nombre de départ à la ligne 70 et le nombre d'arrêt à 128 à la ligne 90.

EXPLICATION DES FORMES D'ONDE

Une note se compose d'une onde sinusoïdale oscillant à la fréquence fondamentale et aux harmoniques de cette onde.

La fréquence fondamentale définit la hauteur globale de la note. Les harmoniques sont des ondes sinusoïdales dont les fréquences sont des multiples entiers de la fondamentale. Une onde sonore se compose de la fréquence fondamentale et de toutes les harmoniques nécessaires pour constituer le son.



En théorie musicale, on suppose que la fréquence fondamentale est l'harmonique numéro 1. La deuxième harmonique a une fréquence double de la fondamentale. La fréquence de la troisième harmonique est le triple de la fondamentale, etc. Le timbre d'une note dépend du nombre d'harmoniques qu'elle contient.

Un instrument de musique, comme une guitare ou un violon, possède une structure harmonique très complexe. En fait, la structure harmonique peut varier pendant la reproduction d'une seule note. Nous avons déjà reproduit les formes d'onde permises par le synthétiseur de musique du Commodore. Voyons maintenant les effets des harmoniques avec les ondes triangulaires, rectangulaires et en dents de scie.

Une onde triangulaire ne contient que des harmoniques impaires. La quantité de chaque harmonique présente est proportionnelle à l'inverse du carré du numéro de l'harmonique. L'harmonique numéro 3 est ainsi atténuée de 1/9 par rapport à l'harmonique numéro 1, car l'harmonique 3 au carré donne 9 (3 X 3) et l'inverse de 9 est 1/9.

On peut voir qu'il existe une analogie de forme entre une onde triangulaire et une onde sinusoïdale oscillant à la fréquence fondamentale.

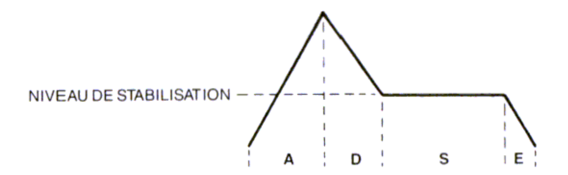
Les ondes en dents de scie contiennent toutes les harmoniques. La quantité de chaque harmonique présente est proportionnelle à l'inverse du numéro de l'harmonique. Par exemple, l'harmonique numéro 2 est moitié moins intense que l'harmonique numéro 1.

L'onde carrée contient des harmoniques impaires proportionnelles à l'inverse du numéro de l'harmonique. Les autres ondes rectangulaires ont un contenu en harmoniques variable. En changeant la largeur de l'impulsion, on peut faire varier considérablement le timbre sonore d'une onde rectangulaire.

Si l'on choisit soigneusement la forme d'onde, on peut commencer avec une structure harmonique ressemblant de près au son désiré. Pour raffiner ce son, on peut ajouter le filtrage qui est un autre aspect de la qualité sonore permise par le Commodore 64. Nous étudierons le *filtrage* dans la suite de cette section.

GÉNÉRATEUR D'ENVELOPPE

Le volume d'une tonalité musicale évolue, dès que l'on commence à l'entendre jusqu'à ce qu'elle s'estompe et qu'elle devienne inaudible. Quand on joue une note, elle monte d'un volume nul à un volume de crête. Cette partie de la note correspond à "**L'ATTAQUE**". La note descend ensuite de son volume de crête à un niveau intermédiaire; cette partie correspond à la **DÉCROISSANCE.** La partie correspondante au niveau intermédiaire proprement dit s'appelle la **STABILISATION.** Enfin, quand la note s'estompe, elle descend du niveau de stabilisation au niveau 0; cette partie correspond à **l'EXTINCTION.** Voici un croquis des quatre phases d'une note:



Chacune des parties ci-dessus donne certaines qualités et limitations à une note. Les limites sont des paramètres.

Les paramètres d'attaque/décroissance/stabilisation/extinction se contrôlent par l'utilisation d'un autre jeu de positions dans la microplaquette de générateur de son. Charger (LOAD) de nouveau le premier exemple de programme. L'exécuter (RUN) et essayer de se rappeler de sa sonorité. Changer ensuite la ligne 20 pour qu'on ait:

EXEMPLE DE PROGRAMME 4 (EXEMPLE 1 MODIFIÉ):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,88:POKES+6,195
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Les registres 5 et 6 définissent l'attaque/décroissance/stabilisation/extinction de la voix 1. L'attaque correspond au demi-octet du registre 5, c'est-à-dire aux quatre positions ou bits de marche/arrêt de poids faible ou fort dans chaque registre. La décroissance correspond au demi-octet de poids faible. On peut choisir un nombre de 0 à 15 pour l'attaque, le multiplier par 16 et l'ajouter à un nombre de 0 à 15 pour la décroissance. Nous donnons ci-dessous les valeurs correspondant à ces nombres.

Le niveau de stabilisation correspond au demi-octet de poids fort du registre 6. Il peut varier de 0 à 15. Il définit la proportion du volume de crête qui correspond au niveau de stabilisation. Le régime d'extinction est défini par le demi-octet de poids faible du registre 6.

Nous donnons ci-dessous les correspondances des valeurs d'attaque, de décroissance et d'extinction:

VALEUR	RÉGIME D'ATTAQUE (TEMPS/CYCLE)	RÉGIME DE DÉCROISSANCE/EXTINCTION (TEMPS/CYCLE)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Essayer les quelques réglages suivants dans l'exemple de programme. On peut aussi en préparer quelques-uns soi-même. La variété des sons possibles est étonnante! Pour obtenir un son de violon, changer la ligne 20 pour qu'on lise:

```
20 POKES+5,88:POKES+6,89:REM A=5;D=8;S=5;R=9
```

Passer à la forme d'onde triangulaire pour avoir un son de xylophone avec les lignes suivantes:

```
20 POKES+5,9:POKES+6,9:REM A=0;D=9;S=0;R=9
70 POKES+4,17
90 POKES+4,16: FORT=1TO50:NEXT
```

Passer à la forme d'onde carrée pour avoir un son de piano avec les lignes suivantes:

```
15 POKES+3,8:POKES+2,0
20 POKES+5,9:POKES+6,0: REM A=0;D=9;S=0;R=0
70 POKES+4,65
90 POKES+4,64:FORT=1TO50:NEXT
```

Les sons les plus intéressants sont ceux particuliers au synthétiseur proprement dit; ce sont des sons qui n'imitent aucun instrument de musique. Essayer par exemple:

```
20 POKES+5,144:POKES+6,243:REM A=9;D=0; S=15; R=3
```

FILTRAGE

On peut changer le contenu en harmoniques d'une forme d'onde à l'aide d'un filtre. La microplaquette SID est dotée de trois types de filtre. On peut les utiliser séparément ou en combinaison. Revenons à l'exemple de programme du début. Nous allons l'utiliser avec un filtre. Nous devons régler plusieurs commandes de filtre.

Ajouter la ligne 15 au programme pour fixer la fréquence de coupure du filtre. Cette fréquence correspond au point de référence du filtre. Les points de coupure de haute et de basse fréquence se fixent dans les registres 21 et 22. Pour mettre en fonction le filtre de la voix; écrire (POKE) dans le registre 23.

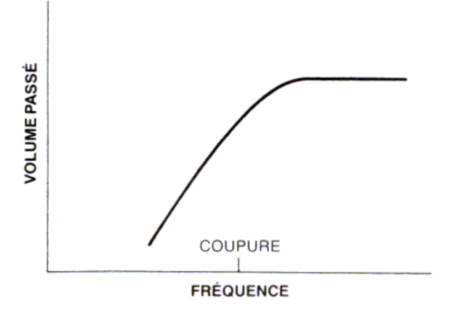
Changer ensuite la ligne 30 pour indiquer qu'on utilise un filtre passe-haut (voir la topographie de registres SID).

EXEMPLE DE PROGRAMME 5 (EXEMPLE 1 MODIFIÉ):

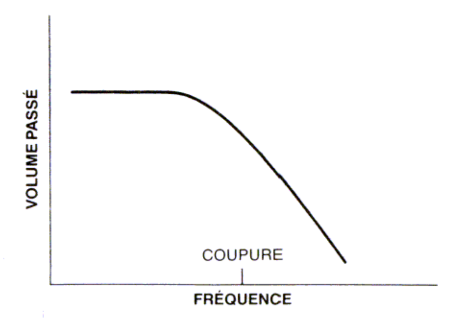
```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
15 POKES+22,128:POKES+21,0:POKES+23,1
20 POKES+5,9:POKES+6,0
30 POKES+24,79
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Exécuter (RUN) ce programme. On peut remarquer que le volume des tonalités plus basses est réduit. Dans sa qualité d'ensemble, la note paraît grêle. En effet, on utilise un filtre passe-haut qui atténue les fréquences au-dessous de la fréquence de coupure spécifiée.

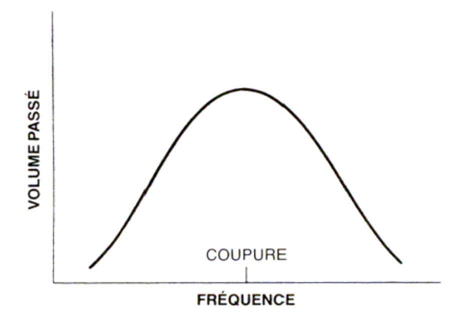
La microplaquette SID de l'ordinateur Commodore possède trois types de filtre. Nous avons utilisé le filtre *passe-haut* qui laisse passer toutes les fréquences égales ou supérieures au point de coupure; il atténue toutes les fréquences au-dessous de la coupure.



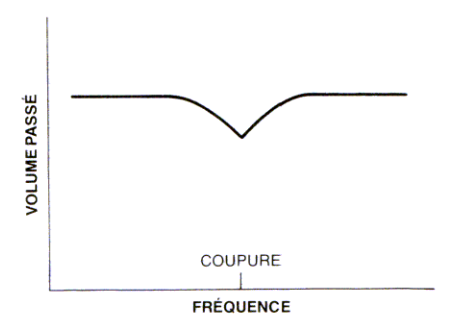
La microplaquette SID possède aussi un filtre *passe-bas.* Ce filtre laisse passer les fréquences au-dessous de la coupure et atténue les fréquences supérieures à cette coupure.



Enfin, la microplaquette est équipée d'un filtre de *bande passante* qui laisse passer une bande étroite de fréquences de chaque côté de la coupure et atténue toutes les autres.



On peut combiner les filtres passe-haut et passe-bas pour former un filtre *éliminateur à flancs raides* qui laisse passer les fréquences écartées de la coupure tout en atténuant la fréquence de coupure proprement dite.



Le registre 24 détermine le type de filtre à utiliser. Ce registre remplit aussi la fonction de commande du volume global. Le bit 6 commande le filtre passe-haut (0 = arrêt, 1 = marche). Le bit 5 correspond au filtre de bande passante, et le bit 4 au filtre passe-bas. Les 3 bits de poids faible de la fréquence de coupure sont déterminés par le registre 21 (L_{fc}) (L_{fc} = 0 à 7). Les 8 bits de la fréquence de coupure haute sont déterminés par le registre 22 (H_{fc}) (H_{fc} = 0 à 255).

Une utilisation avisée du filtrage permet de changer la structure harmonique de toute forme d'onde pour arriver au son désiré. En outre, si l'on change le filtrage d'un son dans ses phases d'attaque/décroissance/stabilisation/extinction, on peut arriver à des effets intéressants.

TECHNIQUES ÉVOLUÉES

On peut changer dynamiquement les paramètres de la microplaquette SID pendant une note ou un son pour créer des effets intéressants et amusants. Pour faciliter les choses, on dispose des sorties numérisées de *l'oscillateur trois et du générateur d'enveloppe trois,* placées respectivement dans les registres 27 et 28.

La sortie de l'oscillateur 3 (registre 27) se rapporte directement à la forme d'onde choisie. Si l'on choisit la forme d'onde en dents de scie de l'oscillateur 3, ce registre présente une série de nombres en progression de 0 à 255 à un régime déterminé par la fréquence de l'oscillateur 3. Si l'on choisit la formule d'onde triangulaire, la sortie augmente de 0 à 255 puis baisse progressivement à 0. Si l'on choisit l'impulsion, la sortie saute de 0 à 255 et vice versa. Enfin, si l'on choisit la forme d'onde de bruit, on obtient une série de nombres au hasard. Si l'on utilise l'oscillateur 3 pour la modulation, on ne désire généralement PAS entendre sa sortie. Pour couper la sortie son de la voix 3, régler le bit 7 du registre 24. Le registre 27 reflète toujours la sortie changeante de l'oscillateur; il n'est absolument pas affecté par le générateur d'enveloppe.

Le registre 25 permet d'accéder à la sortie du générateur d'enveloppe de l'oscillateur 3. Il fonctionne de façon analogue à la sortie de l'oscillateur 3. On doit mettre l'oscillateur en marche pour obtenir une sortie de ce registre.

On peut obtenir un effet de vibrato (variation rapide de fréquence) en ajoutant la sortie de l'oscillateur 3 à la fréquence d'un autre oscillateur. L'exemple de programme 6 en donne une idée.

EXEMPLE DE PROGRAMME 6:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+3,8
40 POKES+5,41:POKES+6,89
50 POKES+14,117
60 POKES+18,16
70 POKES+24,143
80 READFR,DR
90 IFFR=0THENEND
100 POKES+4,65
110 FORT=1TODR*2
120 FQ=FR+PEEK(S+27)/2
130 HF=INT(FQ/256):LF=FQAND255
140 POKES+0,LF:POKES+1,HF
150 NEXT
160 POKES+4,64
170 GOTO80
500 DATA4817,2,5103,2,5407,2
510 DATA8583,4,5407,2,8583,4
520 DATA5407,4,8583,12,9634,2
530 DATA10207,2,10814,2,8583,2
540 DATA9634,4,10814,2,8583,2
550 DATA9634,4,8583,12
560 DATA0,0
```

Nous donnons ci-dessous une explication ligne par ligne de l'exemple de programme 6:

EXPLICATION LIGNE PAR LIGNE DE L'EXEMPLE DE PROGRAMME 6:

Ligne(s)	Description
10	Fixe S au début de la microplaquette de son.
20	Efface toutes les positions de la microplaquette de son.
30	Fixe la largeur d'impulsion haute pour la voix 1.
40	Fixe l'attaque/décroissance pour la voix 1 (A=2, D=9). Fixe la stabilisation/extinction pour la voix 1 (S=5, R=9).
50	Fixe la basse fréquence pour la voix 3.
60	Fixe la forme d'onde triangulaire pour la voix 3.
70	Fixe le volume à 15 et coupe la sortie son pour la voix 3.
80	Lit la fréquence et la durée de la note.
90	Arrête si la fréquence est égale à zéro.
100	Écrit le début de la commande de forme d'onde d'impulsion de la voix 1.
110	Commence la boucle de minutage de durée.
120	Extrait la nouvelle fréquence à l'aide de la sortie de l'oscillateur 3.
130	Extrait les fréquences haute et basse.
140	Écrit les fréquences haute et basse pour la voix 1.
150	Fin de la boucle de minutage.
160	Écrit l'arrêt de la commande de forme d'onde d'impulsion pour la voix 1.
170	Revient à la note suivante.
500-550	Fréquences et durées du chant.
560	Zéros pour signaler la fin du chant.

On peut aussi arriver à une grande variété de créations sonores à l'aide d'effets dynamiques. Par exemple, le programme suivant de sirène change dynamiquement la sortie de fréquence de l'oscillateur 1 quand elle se base sur la sortie de l'onde triangulaire de l'oscillateur 3:

EXEMPLE DE PROGRAMME 7:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+14,5
40 POKES+18,16
50 POKES+3,1
60 POKES+24,143
70 POKES+6,240
80 POKES+4,65
90 FR=5389
100 FORT=1TO200
110 FQ=FR+PEEK(S+27)*3.5
120 HF=INT(FQ/256):LF=FQ-HF*256
130 POKES+0,LF:POKES+1,HF
140 NEXT
150 POKES+24,0
```

Nous donnons ci-dessous une explication ligne par ligne de l'exemple de programme 7:

EXPLICATION LIGNE PAR LIGNE DE L'EXEMPLE DE PROGRAMME 7:

Ligne(s)	Description
10	Fixe S au début de la microplaquette de son.
20	Efface les registres de la microplaquette de son.
30	Fixe la fréquence basse de la voix 3.
40	Fixe la forme d'onde triangulaire pour la voix 3.
50	Fixe la largeur d'impulsion haute pour la voix 1.
60	Fixe le volume à 15 et coupe la sortie son de la voix 3.
70	FIxe la stabilisation/extinction pour la voix 1 (S=15, R=0).
80	Écrit le début de commande de forme d'impulsion de la voix 1.
90	Fixe la fréquence la plus basse de la sirène.
100	Commence la boucle de minutage.
110	Extrait la nouvelle fréquence à l'aide de la sortie de l'oscillateur 3.
120	Extrait les fréquences haute et basse.
130	Écrit les fréquences haute et basse pour la voix 1.
140	Fin de la boucle de minutage.
150	Coupe le volume.

On peut utiliser la forme d'onde de bruit pour obtenir une gamme étendue d'effets sonores. L'exemple suivant imite des applaudissements à l'aide d'une forme d'onde de bruit filtrée.

EXEMPLE DE PROGRAMME 8:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,240:POKES+1,33
40 POKES+5,8
50 POKES+22,104
60 POKES+23,1
70 POKES+24,79
80 FORN=1TO15
90 POKES+4,129
100 FORT=1TO250:NEXT:POKES+4,128
110 FORT=1TO30:NEXT:NEXT
120 POKES+24,0
```

Nous donnons ci-dessous une explication ligne par ligne de l'exemple de programme 8:

EXPLICATION LIGNE PAR LIGNE DE L'EXEMPLE DE PROGRAMME 8:

Ligne(s)	Description
10	Fixe S au début de la microplaquette de son.
20	Efface tous les registres de la microplaquette de son.
30	Fixe les fréquences haute et basse pour la voix 1.
40	Fixe l'attaque/décroissance pour la voix 1 (A=0, D=8).
50	Fixe la fréquence de coupure haute du filtre.
60	Met en marche le filtre pour la voix 1.
70	Fixe le volume à 15 et met le filtre passe-haut en fonction.
80	Compte 15 applaudissements
90	Fixe le début de la commande de forme d'onde de bruit.
100	Attend puis fixe l'arrêt de la commande de forme d'onde de bruit.
110	Attend puis commence l'applaudissement suivant.
120	Coupe le volume.

SYNCHRONISATION ET MODULATION DIRECTIONNELLE

La microplaquette SID 6581 permet de créer des structures harmoniques plus complexes par la synchronisation ou la modulation directionnelle de deux voix.

Le processus de synchronisation se ramène essentiellement à une opération logique ET sur deux formes d'onde. Quand l'une et l'autre sont à zéro, la sortie est également à zéro. L'exemple suivant applique ce processus pour créer une imitation de moustique:

EXEMPLE DE PROGRAMME 9:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,100
40 POKES+5,219
50 POKES+15,28
60 POKES+24,15
70 POKES+4,19
80 FORT=1TO5000:NEXT
90 POKES+4,18
100 FORT=1TO1000:NEXT:POKES+24,0
```

Nous donnons ci-dessous une explication ligne par ligne de l'exemple de programme 9:

EXPLICATION LIGNE PAR LIGNE DE L'EXEMPLE DE PROGRAMME 9:

Ligne(s)	Description
10	Fixe S au début de la microplaquette de son.
20	Efface les registres de la microplaquette de son.
30	Fixe la fréquence haute de la voix 1.
40	Fixe l'attaque/décroissance pour la voix 1 (A=13, D=11).
50	Fixe la fréquence haute de la voix 3.
60	Fixe le volume à 15.
70	Fixe le début de la forme d'onde triangulaire synchronisée pour la voix 1.
80	Boucle de minutage.
90	Fixe l'arrêt de la commande de forme d'onde triangulaire synchronisée pour la voix 1.
100	Attend puis coupe le volume.

La caractéristique de synchronisation est validée (mise en fonction) à la ligne 70 où sont fixés les bits 0, 1 et 4 du registre 4. Le bit 1 valide la fonction de synchronisation entre les voix 1 et 3. Les bits 0 et 4 remplissent leurs fonctions habituelles de déclenchement de la voix 1 et de réglage de la forme d'onde triangulaire.

La modulation directionnelle (obtenue pour la voix 1 en réglant à 1 le bit 3 du registre 4 à la ligne 70 du programme ci-dessous) remplace la sortie triangulaire de l'oscillateur 1 par une combinaison "à modulation directionnelle" des oscillateurs 1 et 3. On obtient ainsi des structures non harmoniques qui servent à imiter des sons de gong ou de cloche. Le programme suivant imite un carillon d'horloge:

EXEMPLE DE PROGRAMME 10:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,130
40 POKES+5,9
50 POKES+15,30
60 POKES+24,15
70 FORL=1TO12:POKES+4,21
80 FORT=1TO1000:NEXT:POKES+4,20
90 FORT=1TO1000:NEXT:NEXT
```

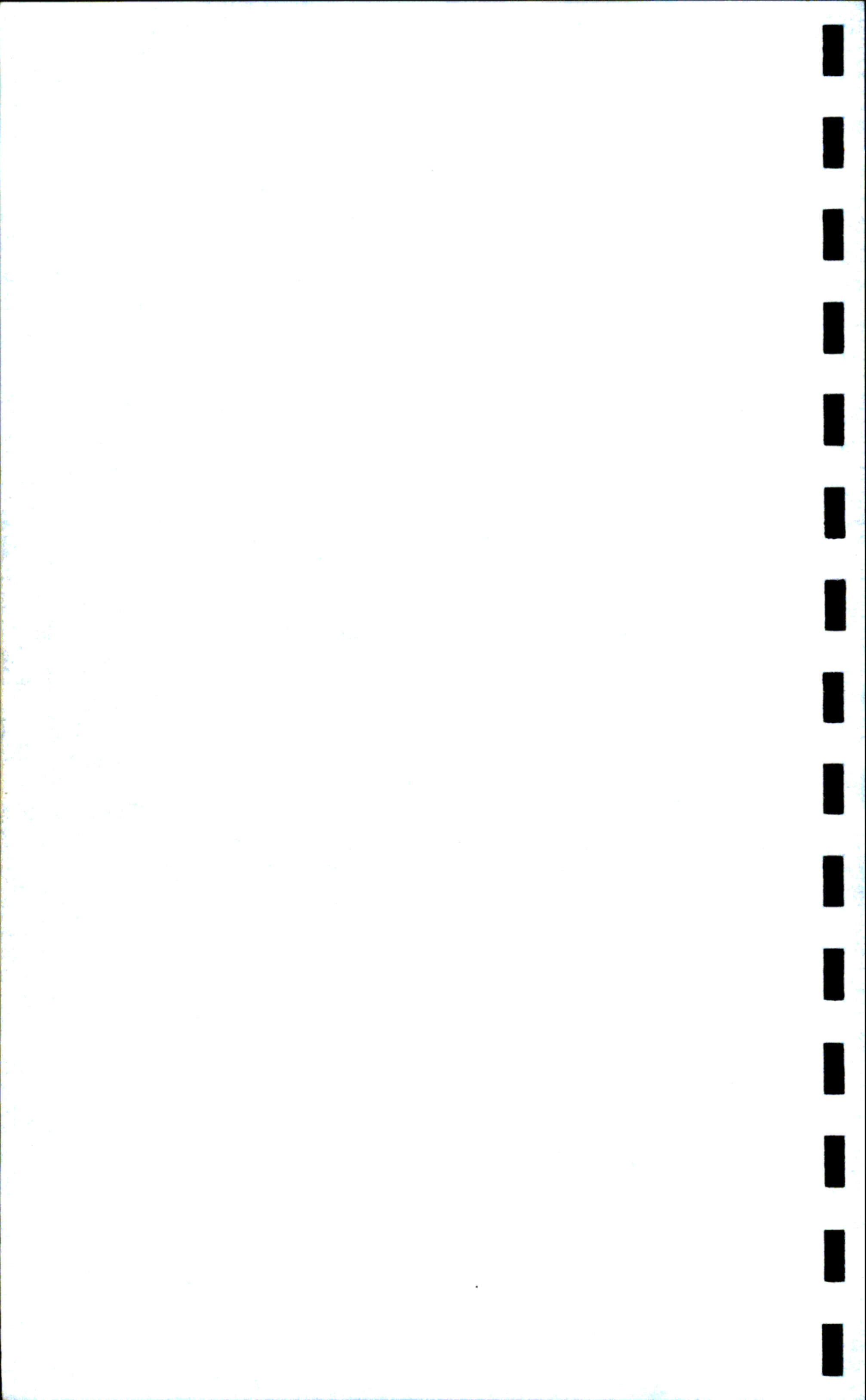
Nous donnons ci-dessous une explication ligne par ligne de l'exemple de programme 10:

EXPLICATION LIGNE PAR LIGNE DE L'EXEMPLE DE PROGRAMME 10:

Ligne(s)	Description
10	Fixe S au début de la microplaquette de son.
20	Efface les registres de la microplaquette de son.
30	Fixe la fréquence haute pour la voix 1.
40	Fixe l'attaque/décroissance pour la voix 1 (A=0, D=9).
50	Fixe la fréquence haute pour la voix 3.
60	Fixe le volume à 15.
70	Compte le nombre de tintements, fixe le début de la commande de forme d'onde triangulaire modulée en direction pour la voix 1.
80	Boucle de minutage; fixe l'arrêt de la modulation directionnelle de l'onde triangulaire.
90	Boucle de minutage; passe au tintement suivant.

Les paramètres de la microplaquette SID du Commodore 64 permettent de réaliser de nombreux effets variés. Seule l'expérience personnelle permet de tirer pleinement parti des possibilités de la machine. Les exemples de cette section du guide de référence ne font qu'effleurer ces possibilités.

Consulter l'ouvrage "**MAKING MUSIC ON YOUR COMMODORE COMPUTER**" qui va de jeux simples et amusants à une instruction musicale de niveau élevé.



CHAPITRE 5

DU BASIC AU LANGAGE MACHINE

- Définition du langage machine
- Programmes en langage machine
- Numération hexadécimale
- Modes d'adressage
- Indexation
- Sous-programmes
- Conseils pour le débutant
- Dispositions pour une tâche importante
- Jeu d'instructions du microprocesseur MCS6510
- Gestion de mémoire avec le Commodore 64
- Le KERNAL
- Opérations de mise sous tension du KERNAL
- Utilisation du langage machine à partir du BASIC
- Topographie de mémoire du Commodore 64

DÉFINITION DU LANGAGE MACHINE

Un microprocesseur central sur microplaquette constitue le "centre nerveux" de tout micro-ordinateur. Le Commodore 64 ne fait pas exception. Chaque microprocesseur comprend son propre langage composé "d'instructions en langage machine". Pour être tout à fait précis, le langage machine est LE SEUL langage de programmation compris par le Commodore 64. Ce langage est la langue "maternelle" de la machine.

Si le langage machine est le seul compris par le Commodore 64, comment peut-on donc lui faire accepter le langage de programmation BASIC CBM? Le BASIC CBM N'EST PAS le langage machine du Commodore 64. Comment le Commodore 64 peut-il donc comprendre des instructions en BASIC CBM comme PRINT et GOTO?

Pour répondre à cette question, il faut d'abord comprendre ce qui se passe dans le Commodore 64. En dehors du microprocesseur, centre nerveux de l'ordinateur, un programme en langage machine est stocké dans une mémoire spéciale que l'on ne peut pas changer. Ce programme ne disparaît pas quand on arrête le Commodore 64, au contraire d'un programme que l'on peut avoir écrit soi-même. Le programme en langage machine est le système d'exploitation du Commodore 64. Le Commodore 64 connaît la marche à suivre quand on le met sous tension, car son système d'exploitation (programme) est automatiquement exécuté (RUN).

Le système d'exploitation doit organiser toute la mémoire de la machine pour différentes tâches. Il s'occupe aussi des caractères tapés au clavier et les affiche sur l'écran; il se charge aussi de nombreuses autres fonctions. Le système d'exploitation constitue "l'intelligence et la personnalité" du Commodore 64 (comme de tout autre ordinateur). Quand on met le Commodore 64 en marche, le système d'exploitation prend les choses en main et, après avoir exécuté les opérations de servitude, il déclare:

READY.
■

Le système d'exploitation du Commodore 64 permet alors la frappe au clavier et l'utilisation de l'éditeur d'écran intégré. L'éditeur d'écran permet de déplacer le curseur, de supprimer, d'insérer des corrections, etc.; ce n'est en fait qu'une des parties du système d'exploitation intégré.

Toutes les commandes du BASIC CMB sont comprises par un autre programme énorme en langage machine incorporé au Commodore 64. Ce programme exécute (RUN) la partie nécessaire de langage machine dépendant de la commande BASIC CBM que l'on passe dans la machine. Ce programme ou interpréteur BASIC interprète les commandes les unes après les autres, jusqu'à ce qu'il en rencontre une qu'il ne comprend pas; le message bien connu apparaît alors:

?SYNTAX ERROR

READY.
■

ASPECT DU CODE MACHINE

On doit se familiariser avec les commandes PEEK et POKE en langage BASIC CMB pour le changement des positions de mémoire. On les a probablement déjà utilisées pour des graphiques sur l'écran et pour les effets sonores. Chaque position de mémoire possède un numéro ou "adresse" qui l'identifie. Si l'on se représente la mémoire du Commodore 64 sous la forme d'une rue avec des édifices, le numéro de chaque porte correspond alors à une adresse. Voyons maintenant les différents types d'utilisation des "édifices" de la rue.

TOPOGRAPHIE SIMPLE DE MÉMOIRE DU COMMODORE 64

ADRESSE	DESCRIPTION
0 & 1	—Registres 6510.
2 à 1023	—Début de la mémoire. —Mémoire utilisée par le système d'exploitation.
1024 à 2039	—Mémoire d'écran.
2040 à 2047	—Pointeurs de caractères graphiques programmables.
2048 à 40959	—Mémoire à la disposition de l'utilisateur. Sert à stocker les programmes en BASIC ou en langage machine.
40960 à 49151	—Interpréteur BASIC CBM de 8 K.
49152 à 53247	—Zone de mémoire vive de programmes spéciaux.
53248 à 53294	—VIC-II.
54272 à 55295	—Registres SID.
55296 à 56296	—Mémoire vive (RAM) couleur.
56320 à 57343	—Registres d'entrée/sortie (6526).
57344 à 65535	—Système d'exploitation KERNAL de CMB, 8 K.

Si l'on ne comprend pas pour le moment la description de chaque partie de la mémoire, les choses s'éclairciront à la lecture des autres sections de ce manuel.

Les programmes en langage machine se composent d'instructions avec ou sans facteurs (paramètres) qui leur sont rattachés. Chaque instruction occupe une position de mémoire; chaque facteur est logé dans une ou deux positions, à la suite de l'instruction.

Dans les programmes en BASIC, les PRINT et GOTO n'occupent en fait qu'une position de mémoire, au lieu d'une pour chaque caractère. On appelle *symbole* le contenu de la position qui correspond à un mot clé particulier en BASIC. En langage machine, il existe différents symboles correspondant à des instructions qui prennent aussi un seul octet (octet = position de mémoire).

Les instructions en langage machine sont très simples. De ce fait, chacune d'elles ne peut accomplir qu'un travail limité. Les instructions en langage machine peuvent changer le contenu d'une position de mémoire ou un des registres internes (positions spéciales de stockage) dans le microprocesseur. Les registres internes forment la base du langage machine.

REGISTRES INTERNES DU MICROPROCESSEUR 6510

ACCUMULATEUR

Ce registre est le plus important du microprocesseur. Différentes instructions de langage machine permettent de copier le contenu d'une position de mémoire dans l'accumulateur, de copier le contenu de l'accumulateur dans une position de mémoire, de modifier directement le contenu de l'accumulateur ou d'un autre registre, sans affecter la mémoire. L'accumulateur est le seul registre possédant les instructions de calcul mathématique.

REGISTRE D'INDEX X

Ce registre très important contient des instructions pour la quasi-totalité des transformations pouvant être apportées à l'accumulateur. Il contient aussi d'autres instructions d'opérations ne pouvant être exécutées que par le registre X. Différentes instructions en langage machine permettent de copier le contenu d'une position de mémoire dans le registre X, de copier le contenu du registre X dans une position de mémoire et de modifier directement le contenu du registre X ou d'un autre registre.

REGISTRE D'INDEX Y

Ce registre très important contient des instructions pour la quasi-totalité des transformations que l'on peut apporter à l'accumulateur et au registre X. Il contient aussi d'autres instructions d'opérations ne pouvant être exécutées que par le registre Y. Différentes instructions en langage machine permettent de copier le contenu d'une position de mémoire dans le registre Y, de copier le contenu du registre Y dans une position de mémoire et de modifier directement le contenu du registre Y ou d'un autre registre.

REGISTRE D'ÉTAT

Ce registre contient huit indicateurs (un indicateur précise si une action s'est produite ou non).

COMPTEUR DE PROGRAMME

Le compteur contient l'adresse de l'instruction en langage machine en cours d'exécution. Le système d'exploitation étant continuellement en cours d'exécution dans le Commodore 64 (il en est de même dans tout autre ordinateur), le compteur de programme change sans cesse. On ne peut l'arrêter qu'en arrêtant le microprocesseur d'une manière quelconque.

POINTEUR DE PILE

Ce registre contient la position de la première place vide de la pile. Les programmes en langage machine et l'ordinateur utilisent la pile pour le stockage temporaire.

ACCÈS D'ENTRÉE/SORTIE

Ce registre apparaît aux positions de mémoire 0 (registre de direction de données) et 1 (accès réel). L'accès d'entrée/sortie a 8 bits. Dans le Commodore 64, ce registre sert à la gestion de mémoire qui permet à la microplaquette de contrôler plus de 64 K de mémoire vive (RAM) et morte (ROM).

Nous ne passons pas ces registres en revue ici. Nous les expliquons à mesure que nous abordons les principes nécessaires qui s'y rattachent.

PROGRAMMES EN LANGAGE MACHINE

Les programmes en langage machine résident en mémoire. Il n'existe aucun dispositif dans le Commodore 64 pour l'écriture et l'édition des programmes en langage machine; on doit donc utiliser un programme à cet effet ou écrire soi-même un programme BASIC qui permette d'écrire en langage machine.

Les programmes *assembleurs* sont le plus couramment utilisés pour l'écriture des programmes en langage machine. Ces logiciels permettent d'écrire des instructions en langage machine sous un format *mnémonique* normalisé qui rend le programme en langage machine plus lisible que s'il se composait d'une suite de nombres. Pour nous résumer, un programme qui permet d'écrire des programmes en langage machine sous format mnémonique est un *assembleur*. Mentionnons qu'un programme qui affiche un programme en langage machine sous format mnémonique est un *désassembleur*. Une cartouche de contrôle de langage machine (avec assembleur/désassembleur, etc.) préparée par Commodore est disponible pour le Commodore 64.

CARTOUCHE 64MON

La cartouche 64MON, en vente chez le dépositaire local, est un programme qui permet de s'évader du BASIC CBM pour s'aventurer dans le langage machine. Elle peut afficher le contenu des registres internes du microprocesseur 6510. Elle permet d'afficher des parties de mémoire et de les changer sur l'écran à l'aide de l'éditeur d'écran. Elle comprend aussi un assembleur et un désassembleur intégrés ainsi que de nombreuses autres caractéristiques qui permettent d'écrire et d'éditer facilement des programmes en langage machine. Il n'est pas nécessaire d'utiliser un assembleur pour écrire en langage machine, mais il facilite considérablement la tâche. Si l'on désire écrire des programmes en langage machine, on recommande fortement de se procurer un assembleur quelconque. Sans assembleur, on doit probablement inscrire (POKE) le programme en langage machine en mémoire, opération totalement déconseillée. Les exemples de ce manuel sont dorénavant présentés dans le format de la cartouche 64MON. La quasi-totalité des assembleurs utilisent le même format; par conséquent, nos exemples en langage machine sont très probablement compatibles avec tout autre assembleur. Avant de passer à l'explication des autres caractéristiques de la cartouche 64MON, examinons le système de numération hexadécimale.

NUMÉRATION HEXADÉCIMALE

La plupart des programmes en langage machine utilisent la numération hexadécimale pour les nombres et adresses.

Dans certains assembleurs, on indique les adresses et nombres en numération décimale (base 10), binaire (base 2), octale (base 8) ou hexadécimale (base 16) (souvent abrégée sous la forme "hex"). Ces assembleurs assurent les conversions pour le programmeur.

La numération hexadécimale peut paraître d'un abord un peu rébarbatif, mais un peu d'entraînement suffit à la maîtriser rapidement.

Si on examine les chiffres décimaux (base 10), on peut voir que chaque chiffre se situe dans l'intervalle compris entre 0 et un chiffre égal à la base moins un (c.-à-d. 9). CETTE REMARQUE S'APPLIQUE À TOUTES LES BASES DE NUMÉRATION. En numération binaire (base 2), les nombres se composent des chiffres zéro et un (soit une unité de moins que la base). Dans le même ordre d'idées, les nombres hexadécimaux se composent de chiffres s'étendant de zéro à quinze; il n'existe cependant pas de chiffres simples pour les nombres dix à quinze; à leur place, on utilise les six premières lettres de l'alphabet.

DÉCIMALE	HEXADÉCIMALE	BINAIRE
0	0	00000000
1	1	00000001
2	2	00000010
3	3	00000011
4	4	00000100
5	5	00000101
6	6	00000110
7	7	00000111
8	8	00001000
9	9	00001001
10	A	00001010
11	B	00001011
12	C	00001100
13	D	00001101
14	E	00001110
15	F	00001111
16	10	00010000

Examinons les choses sous un autre angle. Voici un exemple de construction d'un nombre à base 10 (nombre décimal):

Base élevée aux puissances croissantes: . . .	10^3	10^2	10^1	10^0
Égal:	1000	100	10	1
Soit 4569 (base 10)	4	5	6	9

$$= (4\times1000)+(5\times100)+(6\times10)+9$$

Voyons maintenant un exemple de la construction d'un nombre à base 16 (nombre hexadécimal):

Base élevée aux puissances croissantes: . . .	16^3	16^2	16^1	16^0
Égal:	4096	256	16	1
Soit 11D9 (base 16)	1	1	D	9

$$=1\times4096+1\times256+13\times16+9$$

Donc, 4569 (base 10) = 11D9 (base 16)

L'intervalle des positions adressables de mémoire s'étend de 0 à 65535 (comme nous l'avons précédemment vu). Cet intervalle correspond à 0-FFFF en notation hexadécimale.

En général, les nombres hexadécimaux sont précédés d'un signe dollar ($) pour les distinguer des nombres décimaux. Étudions quelques nombres hexadécimaux avec la cartouche 64MON. Affichons le contenu d'une partie de mémoire en tapant:

SYS 8*4096 (ou SYS 12*4096)
B*
PC SR AC XR YR SP
.; 0401 32 04 5E 00 F6 (peuvent être différents)

Si l'on tape ensuite:

.M 0000 0020 (et appuyer sur RETURN).

on obtient des rangées de 9 nombres hexadécimaux. Le premier nombre à 4 chiffres correspond à l'adresse du premier octet de mémoire montré dans cette rangée. Les 8 autres nombres correspondent au contenu réel des positions de mémoire qui commencent à cette adresse de mémoire.

Il est bon d'essayer d'apprendre à "penser" en hexadécimal. Le processus n'est pas si difficile, car il n'est pas utile de s'occuper de reconvertir en numération décimale. Par exemple, on peut très bien se dire qu'une valeur donnée est stockée à $14ED plutôt qu'à 5357.

PREMIÈRE INSTRUCTION EN LANGAGE MACHINE

LDA — CHARGE L'ACCUMULATEUR

En langage d'assemblage de 6510, la mnémonique correspond toujours à trois caractères. LDA représente "charge l'accumulateur avec . . . ". Le contenu à charger dans l'accumulateur est déterminé par les paramètres associés à cette instruction. L'assembleur connaît le symbole représenté par chaque élément mnémonique. Quand il "assemble une instruction", il met simplement en mémoire (à l'adresse spécifiée) le symbole et les paramètres indiqués. Certains assembleurs donnent des messages d'erreur ou des avertissements si l'on essaie d'assembler des éléments en dehors des possibilités de l'assembleur ou du microprocesseur 6510.

Si l'on place un symbole "#" devant le paramètre rattaché à l'instruction, on désire que le registre spécifié dans l'instruction soit chargé avec la "valeur" qui suit le "#". Par exemple:



Cette instruction place $05 (décimal 5) dans le registre d'accumulateur. Pour cette instruction $A9 (symbole de cette instruction particulière dans ce mode), l'assembleur insère l'adresse spécifiée. Il place $05 dans la position suivante, après celle contenant l'instruction ($A9).

Si le paramètre utilisé par une instruction est précédé d'un "#" (autrement dit, le paramètre correspond à une "valeur" plutôt qu'au contenu d'une position de mémoire ou d'un autre registre), on dit que l'instruction est en mode immédiat. Pour éclaircir les choses, établissons une comparaison avec un autre mode:

Si l'on désire placer le contenu de la position de mémoire $102E dans l'accumulateur, on utilise le mode "absolu" d'instruction:

LDA $102E

L'assembleur fait la distinction entre les deux modes, car le dernier n'a pas de "#" précédant le paramètre. Le microprocesseur 6510 fait la différence entre le mode immédiat et le mode absolu de l'instruction LDA, car ils ont des symboles légèrement différents. LDA (immédiat) a $A9 pour symbole et LDA (absolu) utilise $AD.

La mnémonique qui représente une instruction a généralement un sens implicite. Par exemple, si l'on considère l'instruction LDX, peut-on se faire une idée de son rôle?

Si l'on répond par "charge le registre X avec . . . ", on a mérité la couronne de lauriers! Dans le cas contraire, il n'y a pas lieu de s'inquiéter, car l'apprentissage du langage machine demande de la patience et ne saurait s'apprendre en une journée.

On peut imaginer les différents registres internes comme des positions spéciales de mémoire, car ils peuvent aussi contenir un octet d'information. Il n'est pas utile d'expliquer le système de numération binaire (base 2), car il applique aussi les règles que nous avons écrites précédemment pour les systèmes hexadécimal et décimal. Un "bit" est un chiffre binaire et huit bits constituent un octet. Le nombre maximal que l'on peut donc loger dans un octet est le nombre le plus élevé correspondant à un nombre binaire de huit chiffres. Ce nombre est 11111111 (binaire) que est égal à $FF (en hexadécimal) et à 255 (en décimal). On s'est probablement déjà demandé pourquoi l'on ne peut placer que les nombres de 0 à 255 dans une position de mémoire. Si l'on essaie d'inscrire (POKE) 7680,260 (instruction BASIC qui "dit": "Mettre le numéro deux cent soixante dans la position de mémoire sept mille six cent quatre-vingts"), l'interpréteur BASIC sait que l'on ne peut placer que les nombres 0 à 255 dans une position de mémoire: le Commodore 64 répond alors:

?ILLEGAL QUANTITY ERROR (QUANTITÉ INTERDITE)

READY.
■

Si la limite d'un octet est $FF (hexadécimal), comment le paramètre d'adresse dans l'instruction absolue "LDA $102E" se présente-t-il en mémoire? Il s'exprime sous la forme de deux octets (on ne peut naturellement pas le placer dans un seul). Les deux chiffres de droite de l'adresse hexadécimale forment "l'octet de poids faible" de l'adresse; les deux chiffres de gauche forment "l'octet de poids fort".

Dans le 6510, une adresse doit être spécifiée d'abord par son octet de poids faible, puis par son octet de poids fort. L'instruction "LDA $102E" est ainsi représentée en mémoire par les trois valeurs consécutives suivantes:

$AD, $2E, $10

Pour pouvoir écrire notre premier programme, nous n'avons plus maintenant à connaître qu'une seule instruction qui est BRK. Pour une explication complète de cette instruction, se reporter à l'ouvrage "*M.O.S. 6502 Programming Manual*". Pour le moment, il suffit de l'assimiler à l'instruction END en langage machine.

Si nous écrivons un programme avec la cartouche 64MON et plaçons l'instruction BRK à la fin, le programme est alors exécuté et retourne à 64MON à la fin. Il se peut qu'il n'en soit rien si le programme contient une erreur, car il n'atteint alors jamais l'instruction BRK (en BASIC, une instruction END peut également ne jamais être exécutée). Si le Commodore 64 n'avait pas de touche STOP (arrêt), il ne serait pas possible d'interrompre les programmes BASIC.

ÉCRITURE DU PREMIER PROGRAMME

Si l'on a utilisé l'instruction POKE en BASIC pour placer des caractères sur l'écran, on sait que les codes des caractères de POKE sont différents des valeurs des caractères ASCII de CBM. Par exemple, si l'on introduit:

PRINT ASC("A") (et appuyer sur RETURN)

le Commodore 64 répond par:

65

READY.
■

Si l'on veut cependant mettre un "A" sur l'écran avec l'instruction POKE, le code est 1. Appuyer sur:

SHIFT et CLR/HOME pour effacer l'écran.

POKE 1024,1:POKE 55296,14 (et appuyer sur RETURN) (1024 correspond au début de la mémoire d'écran)

Le "P" de l'instruction POKE doit maintenant être un "A".

Essayons cette opération en langage machine. Taper la ligne suivante dans le programme 64MON. (Le curseur doit clignoter à côté d'un ".".)

.A 1400 LDA #$01 (et appuyer sur RETURN)

Le Commodore 64 demande:

```
.A 1400 A9 01          LDA #$01
.A 1402 ■
```

Taper:

.A 1402 STA $0400

(L'instruction STA stocke le contenu de l'accumulateur dans une position spécifiée de mémoire.)

Le Commodore 64 demande:

.A 1405 ■

Taper ensuite:

.A 1405 LDA #$0E
.A 1407 STA $D800
.A 140A BRK

Effacer l'écran et taper:

G 1400

Si l'on ne s'est pas trompé, le G doit devenir "A".

Nous avons maintenant écrit notre premier programme en langage machine. Son but est de stocker le caractère ("A") à la première position de la mémoire d'écran. Après cela, nous devons maintenant passer à l'étude de quelques autres instructions et principes.

MODES D'ADRESSAGE

PAGE ZÉRO

Nous avons déjà vu que les adresses absolues s'expriment avec un octet de poids fort et un octet de poids faible. L'octet de poids fort est souvent dit "*page de mémoire*". Par exemple, l'adresse $1637 est à la page $16 (22) et $0277 est à la page $02 (2). Il existe un mode spécial d'adressage dit "*page zéro*" qui, comme le nom l'indique, est associé avec l'adressage des positions de mémoire dans la page zéro. Ces adresses ont donc TOUJOURS un bit de poids fort de zéro. En mode d'adressage de page zéro, un octet suffit à préciser l'adresse, plutôt que les deux nécessaires avec une adresse absolue. Le mode d'adressage de page zéro indique à l'ordinateur de supposer que l'octet de poids fort de l'adresse est zéro. L'adressage de page zéro peut donc indiquer des positions de mémoire dont les adresses se situent entre $0000 et $00FF. Cette remarque peut ne pas paraître trop importante pour le moment, mais nous aurons bientôt besoin d'appliquer les principes de l'adressage de page zéro.

PILE

Le microprocesseur 6510 est équipé d'une "pile" qui est utilisée par le programmeur et le microprocesseur pour se souvenir momentanément d'une suite d'événements, par exemple. En BASIC, l'instruction GOSUB qui permet au programmeur d'appeler un sous-programme doit se souvenir du point d'où elle est appelée pour que, à l'exécution de l'instruction RETURN dans le sous-programme, l'interpréteur BASIC "sache" où revenir pour continuer l'exécution. Si l'interpréteur BASIC rencontre une instruction GOSUB dans un programme, il "pousse" sa position présente sur la pile avant de passer au sous-programme; quand une instruction RETURN est exécutée, l'interpréteur "extrait" de la pile les informations qui lui indiquent où il se trouvait avant l'appel du sous-programme. L'interpréteur utilise des instructions comme **PHA** , qui place le contenu de l'accumulateur sur la pile, et **PLA** (le contraire), qui extrait une valeur de la pile et la place dans l'accumulateur. On peut aussi pousser et extraire le registre d'état avec les instructions **PHP** et **PLP** respectivement.

La pile, qui a 256 octets de longueur, est à la page 1 de la mémoire. Elle se trouve donc de $0100 à $01FF; elle est disposée à l'envers. Autrement dit, la première position de la pile est à $01FF et la dernière à $0100. Le pointeur de pile est un autre registre du microprocesseur 6510; il indique toujours la position suivante disponible dans la pile. Quand un élément est mis sur la pile, il se place à l'endroit indiqué par le pointeur qui descend à la position suivante (régression). Quand un élément est extrait de la pile, le pointeur de pile est augmenté; l'octet indiqué par le pointeur est placé dans le registre spécifié.

Jusqu'à ce point, nous avons vu les instructions de modes immédiat, absolu et de page zéro. Nous avons également abordé le mode "implicite" sans toutefois en parler. Dans ce mode, l'information est rendue implicite par une instruction proprement dite, c'est-à-dire par les registres, les indicateurs et la mémoire auxquels se rapporte l'instruction. Nous avons vu jusqu'à présent les instructions PHA, PLA, PHP et PLP qui se rapportent respectivement au traitement de la pile et aux registres d'accumulateur et d'état.

REMARQUE: Dorénavant, nous désignerons le registre X par X, l'accumulateur par A, le registre d'index Y par Y, le pointeur de pile par S et l'unité de traitement par P.

INDEXATION

L'indexation joue un très grand rôle dans le fonctionnement du microprocesseur 6510. On peut la définir comme "la création d'une adresse réelle à partir d'une adresse de base à laquelle on ajoute le contenu des registres d'index X ou Y".

Par exemple, si X contient $05 et si le microprocesseur exécute une instruction LDA en "mode indexé X absolu" avec une adresse de base (par exemple $9000), la position réelle qui est alors chargée dans le registre A est $9000 + $05 = $9005. Le format mnémonique d'une instruction indexée absolue est identique à celui d'une instruction absolue, mais un "X" ou un "Y" indique que l'index est ajouté à l'adresse.

EXEMPLE:

LDA $9000,X

Avec le microprocesseur 6510, on dispose des modes d'adressage indexé absolu, indexé de page zéro, indexé indirect et indirect indexé.

MODE INDEXÉ INDIRECT

Ce mode ne permet d'utiliser que le registre Y comme index. L'adresse réelle ne peut être que dans la page zéro. Le mode d'instruction est dit indirect, car l'adresse de page zéro spécifiée dans l'instruction contient l'octet de poids faible de l'adresse réelle; l'octet suivant contient l'octet de poids fort.

EXEMPLE:

Supposons que la position $02 contienne $45 et que la position $03 contienne $1E. Si l'instruction de chargement de l'accumulateur en mode indexé indirect est exécutée et si l'adresse de page zéro spécifiée est $02, l'adresse réelle devient alors:

Octet de poids faible = contenu de $02
Octet de poids fort = contenu de $03
Registre Y = $00

L'adresse réelle est donc: $1E45 + Y = $1E45.

Par son titre, ce mode implique en fait un principe indirect qui peut être difficile à saisir au premier abord. Examinons les choses sous un autre angle:

"Je vais porter cette lettre à la poste. L'adresse est $02,RUE DE LA MÉMOIRE; l'adresse de la lettre est à $05 maisons après $1600, RUE DE LA MÉMOIRE." En code, on obtient les équivalents suivants:

```
LDA #$00      —charge l'adresse de base réelle d'octet de poids faible
STA $02       —fixe l'octet de poids faible de l'adresse indirecte
LDA #$16      —charge l'adresse indirecte de l'octet de poids fort
STA $03       —fixe l'octet de poids fort de l'adresse indirecte
LDY #$05      —fixe l'index indirect (Y)
LDA ($02),Y   —charge indirectement indexée par Y
```

MODE INDIRECT INDEXÉ

Le mode indirect indexé permet de n'utiliser que le registre X comme index. Il est identique au mode indexé indirect, mais on indexe en fait l'adresse de page zéro du pointeur et non l'adresse de base réelle. Par suite, l'adresse de base réelle EST l'adresse réelle, car l'index a déjà été utilisé en mode indirect. On utiliserait également le mode indirect indexé si une table de pointeurs indirects se trouvait dans la mémoire de page zéro et si le registre X pouvait alors spécifier le pointeur indirect à utiliser.

EXEMPLE:

Supposons que la position $02 contienne $45 et que la position $03 contienne $10. Si l'instruction de charge de l'accumulateur en mode indirect indexé est exécutée et si l'adresse spécifiée de page zéro est $02, l'adresse réelle est alors:

Octet de poids faible = contenu de ($02 + X)
Octet de poids fort = contenu de ($03 + X)
Registre X = $00

Le pointeur réel est ainsi à: $02 + X = $02.

Par conséquent, l'adresse réelle est l'adresse indirecte contenue dans $02 qui est encore $1045.

Par son titre, ce mode implique en fait le principe indirect, mais il peut être difficile à saisir à premier abord. Étudions la chose sous un autre angle:

"Je vais porter cette lettre à la quatrième poste à l'adresse $01 RUE DE LA MÉMOIRE; la lettre porte l'adresse $1600, RUE DE LA MÉMOIRE." En code, on obtient:

```
LDA #$00      —charge l'adresse de base réelle d'octet de poids faible
STA $06       —fixe l'octet de poids faible de l'adresse indirecte
LDA #$16      —charge l'adresse indirecte de l'octet de poids fort
STA $07       —fixe l'octet de poids fort de l'adresse indirecte
LDX #$05      —fixe l'index indirect (X)
LDA ($02,X)   —charge indirectement indexée par X
```

REMARQUE: Des deux modes indirects d'adressage, le premier (mode indexé indirect) est le plus largement utilisé.

BRANCHEMENTS ET VÉRIFICATION

Le langage machine repose sur un autre principe très important qui est la possibilité de vérifier et de déceler certaines conditions, de façon analogue à la structure "IF . . . THEN, IF . . . GOTO" en BASIC CBM.

Les divers *indicateurs* du registre d'état sont affectés de différentes manières par différentes instructions. Par exemple, un indicateur est mis à un quand une instruction a amené un résultat zéro et il est remis à zéro si un résultat n'est pas égal à zéro. L'instruction:

LDA #$00

met *l'indicateur de résultat zéro* à un, car l'instruction a amené un contenu de zéro dans l'accumulateur.

Avec une condition donnée, un jeu d'instructions assure le *branchement* à une autre partie du programme. **BEQ** est un exemple d'instruction de branchement. **BEQ** signifie *branchement si le résultat est égal à zéro (Branch if result EQual to zero)*. Le branchement se fait si la condition est vraie; si ce n'est pas le cas, le programme continue avec l'instruction suivante, comme si rien ne s'était passé. Les instructions de branchement ne se basent pas sur le résultat des instructions précédentes, mais sur l'examen interne du registre d'état. Nous venons de mentionner que le registre d'état comprenait un *indicateur de résultat zéro*. L'instruction BEQ assure le branchement si l'indicateur de résultat zéro (indicateur **Z**) est à un. Chaque instruction de branchement possède une instruction opposée. À BEQ est opposée l'instruction **BNE** qui indique le *branchement si le résultat n'est pas égal à zéro (Branch on result Not Equal to zero)*, c'est-à-dire si *Z* n'est pas à un.

Les registres d'index possèdent plusieurs instructions connexes qui modifient leur contenu. Par exemple, l'instruction **INX** *augmente le registre d'index X*. Si le registre X contient $FF avant d'être augmenté (nombre maximal pouvant être contenu dans le registre X), il revient automatiquement à zéro. Si l'on veut qu'un programme continue une tâche jusqu'à ce qu'on ait procédé à l'augmentation de l'index X qui l'a amené à zéro, on peut utiliser l'instruction BNE pour continuer le "bouclage" jusqu'à ce que X devienne zéro.

DEX, inverse de INX, indique la *diminution du registre d'index X (DEcrement the X index register)*. Si le registre d'index X est à zéro, DEX assure le retour à $FF. De même, le *registre d'index Y* possède les instructions **INY** et **DEY**.

Que se passerait-il donc si un programme ne pouvait pas attendre que X ou Y arrive ou n'arrive pas à zéro? On dispose alors des *instructions de comparaison* **CPX** et **CPY** qui permettent au programmeur en langage machine de contrôler les registres d'index et même le contenu des positions de mémoire avec des valeurs spécifiques. Si l'on désire savoir si le registre X contient $40, on peut utiliser l'instruction:

CPX #$40	**—compare X à la "valeur" $40.**
BEQ (autre partie du programme)	**—assure le branchement à un autre point du programme, si cette condition est "vraie".**

Les instructions de comparaison et de branchement occupent une place importante dans un programme en langage machine.

Quand on utilise le 64MON, le facteur spécifié dans une instruction de branchement correspond à l'adresse de la partie du programme où va le branchement quand les conditions appropriées sont satisfaites. Le facteur n'est cependant qu'un *décalage* qui déplace l'utilisateur du point où se trouve présentement le programme à l'adresse spécifiée. Ce décalage correspond à un octet; par suite, l'intervalle permis par une instruction de branchement est limité. Il permet le branchement de 128 octets en arrière à 127 octets en avant.

REMARQUE: On arrive ainsi à un intervalle total de 255 octets qui correspond naturellement à l'intervalle maximal des valeurs qu'un octet peut contenir.

Le 64MON indique si l'on a fait un "branchement en dehors de l'intervalle" en refusant "d'assembler" l'instruction en cause. Ne pas s'inquiéter de cet inconvénient pour le moment, car il est peu probable que l'on rencontre ce genre de branchement avant quelque temps. En langage machine et par rapport à l'adresse absolue, le branchement est une instruction "rapide", grâce au principe du "décalage". Le 64MON permet de taper une adresse absolue et il calcule le décalage correct. Nous avons là l'un des avantages de l'utilisation d'un assembleur.

REMARQUE: Il n'est PAS possible de voir chaque instruction de branchement. Pour plus de détails, se reporter à la bibliographie de l'annexe F.

SOUS-PROGRAMMES

En langage machine, comme en BASIC, on peut appeler des sous-programmes. L'instruction d'appel d'un sous-programme est **JSR** (saut au sous-programme — Jump to SubRoutine), suivie de l'adresse absolue spécifiée.

Dans le système d'exploitation se trouve un sous-programme en langage machine qui imprime un caractère sur l'écran. Le code ASCII CBM du caractère doit être dans l'accumulateur avant qu'on appelle le sous-programme. $FFD2 est l'adresse de ce sous-programme.

Pour imprimer "HI" sur l'écran, on doit donc introduire le programme suivant:

.A 1400 LDA #$48	**—charge le code ASCII CBM de "H"**
.A 1402 JSR $FFD2	**—et l'imprime**
.A 1405 LDA #$49	**—charge le code ASCII CMB de "I"**
.A 1407 JSR $FFD2	**—et l'imprime aussi**
.A 140A LDA #$0D	**—imprime un retour du chariot**
.A 140C JSR $FFD2	
.A 140F BRK	**—retourne au 64MON**
.G 1400	**—imprime "HI" et retourne au 64MON**

Le sous-programme d'impression (PRINT) de caractère que nous venons d'utiliser fait partie de la *table de saut* de KERNAL. L'instruction analogue à GOTO en BASIC est **JMP** ou *saut à l'adresse absolue spécifiée (JuMP to the specified absolute address)*. Le KERNAL est une longue liste de sous-programmes "normalisés" qui commandent toutes les entrées et sorties du Commodore 64. Chaque entrée dans le KERNAL saute à un sous-programme dans le système d'exploitation. Cette "table de sauts" se trouve entre les positions de mémoire $FF84 et $FFF5 dans le système d'exploitation. On trouvera une explication détaillée du KERNAL dans la section de référence du KERNAL de ce manuel. Nous utilisons cependant ici certains programmes de routine pour montrer la simplicité et l'efficacité du KERNAL.

Mettons maintenant en pratique les nouveaux principes que nous venons de voir dans un autre programme pour montrer l'utilisation pratique des instructions:

Ce programme permet d'afficher l'alphabet avec un programme de routine KERNAL. Nous ne rencontrons ici qu'une seule nouvelle instruction qui est **TXA** *ou transfert du contenu du registre d'index X dans l'accumulateur (Transfer the contents of the X index register into the Accumulator).*

.A 1400 LDX #$41	**—X = ASCII CMB de "A"**
.A 1402 TXA	**—A = X**
.A 1403 JSR $FFD2	**—imprime le caractère**
.A 1406 INX	**—évacue le compte**
.A 1407 CPX #$5B	**—avons-nous dépassé "Z"?**
.A 1409 BNE $1402	**—non, on revient et on continue**
.A 140B BRK	**—oui, on revient au 64MON**

Pour que le Commodore 64 imprime l'alphabet, taper la commande bien connue:

.G 1400

Les remarques qui accompagnent le programme expliquent son déroulement et sa logique. Si l'on écrit un programme, le noter d'abord sur papier, puis le vérifier par petites parties, dans la mesure du possible.

CONSEILS POUR LE DÉBUTANT

Pour mieux se familiariser avec le langage machine, nous recommandons d'étudier les programmes d'autres programmeurs dans ce langage. On rencontre continuellement ces programmes dans les magazines et les bulletins. On peut les étudier, même si l'article se rapporte à un autre ordinateur qui utilise aussi le microprocesseur 6510 ou 6502. Il faut bien comprendre le code utilisé. Il faut faire preuve de persévérance, en particulier quand on rencontre une technique toute nouvelle. On peut parfois se sentir découragé, mais la patience doit conduire au succès.

Quand on s'est familiarisé avec d'autres programmes en langage machine, on DOIT en écrire soi-même. On peut écrire des programmes utilitaires en BASIC ou un programme entièrement en langage machine.

Il faut aussi utiliser les programmes utilitaires disponibles, dans l'ordinateur ou dans un programme. Ils facilitent l'écriture, l'édition ou la recherche des erreurs dans un programme en langage machine. Citons par exemple le KERNAL qui permet de vérifier le clavier, d'imprimer le texte, de contrôler les dispositifs périphériques comme les unités de disque, les imprimantes, les modems, etc. et de gérer la mémoire et l'écran. Le KERNEL est très puissant; nous recommandons fortement de l'utiliser (voir la section sur le KERNAL à la page 270).

L'écriture des programmes en langage machine présente les avantages suivants:

1. Rapidité — Le langage machine est cent fois et parfois des milliers de fois plus rapide qu'un langage de haut niveau comme le BASIC.
2. Caractère hermétique — Un programme en langage machine peut être totalement "hermétique"; on peut amener l'utilisateur à ne faire que ce que le programme permet et rien d'autre. Avec un langage de haut niveau, il faut espérer que l'utilisateur n'arrête pas l'interpréteur BASIC en introduisant, par exemple, un zéro qui amène ensuite:

?DIVISION BY ZERO ERROR IN LINE 830

(ERREUR DE DIVISION PAR ZERO A LA LIGNE 830)

READY.
■

En bref, le programmeur en langage machine peut tirer le maximum des possibilités de l'ordinateur.

DISPOSITIONS POUR UNE TÂCHE IMPORTANTE

Quand on aborde une tâche importante en langage machine, on y a généralement déjà accordé une certaine place dans le subconscient. On peut penser au déroulement de certains processus en langage machine. Quand on commence la tâche, il est général bon d'en noter les étapes sur le papier. Dans ce but, utiliser des schémas d'utilisation de mémoire, des modules fonctionnels des codes requis et un organigramme. Supposons que l'on désire écrire un jeu de roulette en langage machine. On peut décrire ses grandes lignes de la façon suivante:

- Afficher le titre
- Demander si le joueur a besoin d'instructions
- OUI; les afficher; aller au départ
- NON; aller au départ
- Commencer l'initialisation
- Présenter une table de roulette sur l'affichage principal
- Prendre les paris
- Faire tourner la roue
- Faire ralentir la roue jusqu'à l'arrêt
- Comparer les paris au résultat
- Informer le joueur
- Le joueur a-t-il encore de l'argent?
- OUI; aller à l'affichage principal
- NON; aviser le joueur et revenir au départ

Cet aperçu général peut se décomposer davantage quand on aborde chaque module. Si l'on considère une tâche énorme qui semble impossible à résoudre, on peut la décomposer en parties assez petites offrant des solutions qui s'emboîtent ensuite les unes dans les autres.

Ce processus ne s'améliore qu'avec la pratique; il ne faut pas hésiter à se remettre sans cesse à la tâche.

JEU D'INSTRUCTIONS DU MICROPROCESSEUR

ADC	Addition de la mémoire à l'accumulateur avec report
AND	Opération "ET" de la mémoire avec l'accumulateur
ASL	Décalage d'un bit vers la gauche (mémoire ou accumulateur)
BCC	Branchement sur effacement du report
BCS	Branchement sur mise du report à un
BEQ	Branchement sur le résultat zéro
BIT	Vérification des bits en mémoire avec l'accumulateur
BMI	Branchement sur résultat négatif
BNE	Branchement sur résultat non nul
BPL	Branchement sur résultat positif
BRK	Interruption forcée
BVC	Branchement sur effacement de dépassement de capacité
BVS	Branchement sur mise à un du dépassement de capacité
CLC	Effacement de l'indicateur de report
CLD	Effacement du mode décimal
CLI	Effacement du bit d'invalidation d'interruption
CLV	Effacement de l'indicateur de dépassement de capacité
CMP	Comparaison de la mémoire et de l'accumulateur
CPX	Comparaison de la mémoire et de l'index X
CPY	Comparaison de la mémoire et de l'index Y
DEC	Régression de un de la mémoire
DEX	Régression de un de l'index X
DEY	Régression de un de l'index Y
EOR	Opération "Ou exclusif" de la mémoire avec l'accumulateur
INC	Progression de un de la mémoire
INX	Progression de un de l'index X
INY	Progression de un de l'index Y
JMP	Saut à une nouvelle position
JSR	Saut à une nouvelle position avec sauvegarde de l'adresse de retour

MCS6510 — ORDRE ALPHABÉTIQUE

LDA	Chargement de la mémoire dans l'accumulateur
LDX	Chargement de la mémoire dans l'index X
LDY	Chargement de la mémoire dans l'index Y
LSR	Décalage d'un bit vers la droite (mémoire ou accumulateur)
NOP	Pas d'opération
ORA	Opération ''OU'' de la mémoire avec l'accumulateur
PHA	Accumulateur placé sur la pile
PHP	État de l'unité de traitement placé sur la pile
PLA	Accumulateur extrait de la pile
PLP	État de l'unité de traitement extrait de la pile
ROL	Rotation d'un bit vers la gauche (mémoire ou accumulateur)
ROR	Rotation d'un bit vers la droite (mémoire ou accumulateur)
RTI	Retour d'une interruption
RTS	Retour d'un sous-programme
SBC	Soustraction de la mémoire de l'accumulateur avec retenue
SEC	Établissement de l'indicateur de report
SED	Établissement du mode décimal
SEI	Établissement de l'état d'invalidation d'interruption
STA	Stockage de l'accumulateur en mémoire
STX	Stockage de l'index X en mémoire
STY	Stockage de l'index Y en mémoire
TAX	Transfert de l'accumulateur à l'index X
TAY	Transfert de l'accumulateur à l'index Y
TSX	Transfert du pointeur de pile à l'index X
TXA	Transfert de l'index X à l'accumulateur
TXS	Transfert de l'index X au pointeur de pile
TYA	Transfert de l'index Y à l'accumulateur

Nous utilisons la notation ci-après avec ce résumé:

A	Accumulateur
X, Y	Registres d'index
M	Mémoire
P	Registre d'état de l'unité de traitement
S	Pointeur de pile
✓	Changement
—	Pas de changement
+	Addition
∧	ET logique
−	Soustraction
⊻	Ou exclusif logique
↑	Transfert venant de la pile
↓	Transfert vers la pile
→	Transfert vers
←	Transfert venant de
V	OU logique
PC	Compteur de programme
PCH	Compteur haut de programme
PCL	Compteur bas de programme
OPER	Facteur
#	MODE D'ADRESSAGE IMMÉDIAT

Remarque: En haut de chaque table, un numéro de référence (Réf.:XX) est indiqué entre parenthèses. Il dirige l'utilisateur à la section du manuel de programmation des micro-ordinateurs MCS6500 dans lequel l'instruction est définie et étudiée.

ADC *Addition de la mémoire à l'accumulateur avec report* ADC

Opération: A + M + C → A, C

N Ƶ C I D V

(Réf.: 2.2.1) ✓ ✓ ✓ — — ✓

Mode d'adressage	Forme en langage d'assemblage		Code OP	Nbre d'octets	Nbre de cycles
Immédiat	ADC	# Oper	69	2	2
Page zéro	ADC	Oper	65	2	3
Page zéro, X	ADC	Oper, X	75	2	4
Absolu	ADC	Oper	6D	3	4
Absolu, X	ADC	Oper, X	7D	3	4*
Absolu, Y	ADC	Oper, Y	79	3	4*
(Indirect, X)	ADC	(Oper, X)	61	2	6
(Indirect), Y	ADC	(Oper), Y	71	2	5*

* Ajouter 1 si l'on passe la limite de page.

AND *Opération "ET" de la mémoire avec l'accumulateur* AND

Opération ET logique vers l'accumulateur

Opération: A ∧ M → A

N Ƶ C I D V

(Réf.: 2.2.3.0) ✓ ✓ — — — —

Mode d'adressage	Forme en langage d'assemblage		Code OP	Nbre d'octets	Nbre de cycles
Immédiat	AND	# Oper	29	2	2
Page zéro	AND	Oper	25	2	3
Page zéro, X	AND	Oper, X	35	2	4
Absolu	AND	Oper	2D	3	4
Absolu, X	AND	Oper, X	3D	3	4*
Absolu, Y	AND	Oper, Y	39	3	4*
(Indirect, X)	AND	(Oper, X)	21	2	6
(Indirect), Y	AND	(Oper), Y	31	2	5

* Ajouter 1 si l'on passe la limite de page.

ASL **ASL** *Décalage d'un bit vers la gauche (mémoire ou accumulateur)* ASL

Opération: C ← | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ← 0   N Z C I D V

✓ ✓ ✓ — — —

(Réf.: 10.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Accumulateur	ASL A	0A	1	2
Page zéro	ASL Oper	06	2	5
Page zéro, X	ALS Oper, X	16	2	6
Absolu	ALS Oper	0E	3	6
Absolu, X	ALS Oper, X	1E	3	7

BCC **BCC** *Branchement sur effacement du report* BCC

Opération: Branchement sur C = 0   N Z C I D V

— — — — — —

(Réf.: 4.1.1.3)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BCC Oper	90	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit sur une autre page.

BCS **BCS** *Branchement sur mise du report à un* BCS

Opération: Branchement à C = 1   N Z C I D V

— — — — — —

(Réf.: 4.1.1.4)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BCS Oper	B0	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit à la page suivante.

BEQ — **BEQ** *Branchement sur le résultat zéro* — BEQ

Opération: Branchement à Z = 1

N Z C I D V
— — — — — —

(Réf.: 4.1.1.5)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BEQ Oper	F0	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit à la page suivante.

BIT — **BIT** *Vérification des bits en mémoire avec l'accumulateur* — BIT

Opération: A ∧ M, $M_7 \rightarrow N$, $M_6 \rightarrow V$

Les bits 6 et 7 sont transférés au registre d'état.
Si le résultat de A ∧ M est zéro, on a donc Z = 1,
sinon Z = 0

N Z C I D V
M_7 ✓ — — — M_6

(Réf.: 4.2.1.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Page zéro	BIT Oper	24	2	3
Absolu	BIT Oper	2C	3	4

BMI — **BMI** *Branchement sur résultat négatif* — BMI

Opération: Branchement à N = 1

N Z C I D V
— — — — — —

(Réf.: 4.1.1.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BMI Oper	30	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit à une autre page.

BNE

BNE *Branchement sur résultat non nul*

BNE

Opération: Branchement à Z = 0

N Z C I D V

_ _ _ _ _ _

(Réf.: 4.1.1.6)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BNE Oper	D0	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit à une autre page.

BPL

BPL *Branchement sur résultat positif*

BPL

Opération: Branchement à N = 0

N Z C I D V

_ _ _ _ _ _

(Réf.: 4.1.1.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BPL Oper	10	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit sur une autre page.

BRK

BRK *Interruption forcée*

BRK

Opération: Interruption forcée PC + 2 ↓ P ↓

N Z C I D V

_ _ _ 1 _ _

(Réf.: 9.11)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	BRK	00	1	7

1. On ne peut pas masquer une commande BRK en mettant I à 1.

BVC — BVC *Branchement sur effacement de dépassement de capacité* — BVC

Opération: Branchement à V = 0

N Z C I D V
— — — — — —

(Réf.: 4.1.1.8)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BVC Oper	50	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit sur une autre page.

BVS — BVS *Branchement sur mise à un du dépassement de capacité* — BVS

Opération: Branchement à V = 1

N Z C I D V
— — — — — —

(Réf.: 4.1.1.7)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Relatif	BVS Oper	70	2	2*

* Ajouter 1 si le branchement se produit sur la même page.
* Ajouter 2 si le branchement se produit sur une autre page.

CLC — CLC *Effacement de l'indicateur de report* — CLC

Opération: 0 → C

N Z C I D V
— — 0 — — —

(Réf.: 3.0.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	CLC	18	1	2

CLD

CLD *Effacement du mode décimal*

CLD

Opération: Ø → D

N	Z	C	I	D	V
—	—	—	—	Ø	—

(Réf.: 3.3.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	CLD	D8	1	2

CLI

CLI *Effacement du bit d'invalidation d'interruption*

CLI

Opération: Ø → I

N	Z	C	I	D	V
—	—	—	Ø	—	—

(Réf.: 3.2.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	CLI	58	1	2

CLV

CLV *Effacement de l'indicateur de dépassement de capacité*

CLV

Opération: Ø → V

N	Z	C	I	D	V
—	—	—	—	—	Ø

(Réf.: 3.6.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	CLV	B8	1	2

CMP — CMP *Comparaison de la mémoire et de l'accumulateur* — CMP

Opération: A − M

N Z C I D V
✓ ✓ ✓ _ _ _

(Réf.: 4.2.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	CMP #Oper	C9	2	2
Page zéro	CMP Oper	C5	2	3
Page zéro, X	CMP Oper, X	D5	2	4
Absolu	CMP Oper	CD	3	4
Absolu, X	CMP Oper, X	DD	3	4*
Absolu, Y	CMP Oper, Y	D9	3	4*
(Indirect, X)	CMP (Oper, X)	C1	2	6
(Indirect), Y	CMP (Oper), Y	D1	2	5*

* Ajouter 1 si l'on passe la limite de page.

CPX — CPX *Comparaison de la mémoire et de l'index X* — CPX

Opération: X − M

N Z C I D V
✓ ✓ ✓ _ _ _

(Réf.: 7.8)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	CPX #Oper	E0	2	2
Page zéro	CPX Oper	E4	2	3
Absolu	CPX Oper	EC	3	4

CPY — CPY *Comparaison de la mémoire et de l'index Y* — CPY

Opération: Y − M

N Z C I D V
✓ ✓ ✓ _ _ _

(Réf.: 7.9)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	CPY #Oper	C0	2	2
Page zéro	CPY Oper	C4	2	3
Absolu	CPY Oper	CC	3	4

DEC

DEC *Régression de un de la mémoire* **DEC**

Opération: M – 1 → M

N Z C I D V

✓ ✓ — — — —

(Réf.: 10.7)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Page zéro	DEC Oper	C6	2	5
Page zéro, X	DEC Oper, X	D6	2	6
Absolu	DEC Oper	CE	3	6
Absolu, X	DEC Oper, X	DE	3	7

DEX

DEX *Régression de un de l'index X* **DEX**

Opération: X – 1 → X

N Z C I D V

✓ ✓ — — — —

(Réf.: 7.6)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	DEX	CA	1	2

DEY

DEY *Régression de un de l'index Y* **DEY**

Opération: Y – 1 → Y

N Z C I D V

✓ ✓ — — — —

(Réf.: 7.7)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	DEY	88	1	2

EOR **EOR** *Opération "Ou exclusif" de la mémoire avec l'accumulateur* EOR

Opération: A ⊻ M → A

N Z C I D V

✓ ✓ — — — —

(Réf.: 2.2.3.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	EOR #Oper	49	2	2
Page zéro	EOR Oper	45	2	3
Page zéro, X	EOR Oper, X	55	2	4
Absolu	EOR Oper	4D	3	4
Absolu, X	EOR Oper, X	5D	3	4*
Absolu, Y	EOR Oper, Y	59	3	4*
(Indirect, X)	EOR (Oper, X)	41	2	6
(Indirect),Y	EOR (Oper), Y	51	2	5*

* Ajouter 1 si l'on passe la limite de page

INC **INC** *Progression de un de la mémoire* INC

Opération: M + 1 → M

N Z C I D V

✓ ✓ — — — —

(Réf.: 10.6)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Page zéro	INC Oper	E6	2	5
Page zéro, X	INC Oper, X	F6	2	6
Absolu	INC Oper	EE	3	6
Absolu, X	INC Oper, X	FE	3	7

INX **INX** *Progression de un de l'index X* INX

Opération: X + 1 → X

N Z C I D V

✓ ✓ — — — —

(Réf.: 7.4)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	INX	E8	1	2

INY

INY *Progression de un de l'index Y*

Opération: Y + 1 → Y

N Z C I D V
✓ ✓ — — — —

(Réf.: 7.5)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	INY	C8	1	2

JMP

JMP *Saut à une nouvelle position*

Opération: (PC + 1) → PCL
(PC + 2) → PCH

(Réf.: 4.0.2)
(Réf.: 9.8.1)

N Z C I D V
— — — — — —

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Absolu	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

JSR

JSR *Saut à une nouvelle position avec sauvegarde de l'adresse de retour*

Opération: PC + 2 ↓ , (PC + 1) → PCL
(PC + 2) → PCH

N Z C I D V
— — — — — —

(Réf.: 8.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Absolu	JSR Oper	20	3	6

LDA — **LDA** *Chargement de la mémoire dans l'accumulateur* — LDA

Opération: M → A

N	Z	C	I	D	V
✓	✓	—	—	—	—

(Réf.: 2.1.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	LDA #Oper	A9	2	2
Page zéro	LDA Oper	A5	2	3
Page zéro, X	LDA Oper, X	B5	2	4
Absolu	LDA Oper	AD	3	4
Absolu, X	LDA Oper, X	BD	3	4*
Absolu, Y	LDA Oper, Y	B9	3	4*
(Indirect, X)	LDA (Oper, X)	A1	2	6
(Indirect), Y	LDA (Oper), Y	B1	2	5*

* Ajouter 1 si l'on passe la limite de page.

LDX — **LDX** *Chargement de la mémoire dans l'index X* — LDX

Opération: M → X

N	Z	C	I	D	V
✓	✓	—	—	—	—

(Réf.: 7.0)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	LDX #Oper	A2	2	2
Page zéro	LDX Oper	A6	2	3
Page zéro, Y	LDX Oper, Y	B6	2	4
Absolu	LDX Oper	AE	3	4
Absolu, Y	LDX Oper, Y	BE	3	4*

* Ajouter 1 si l'on passe la limite de page.

LDY — LDY *Chargement de la mémoire dans l'index Y* — LDY

Opération: M → Y

N	Z	C	I	D	V
✓	✓	—	—	—	—

(Réf.: 7.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	LDY #Oper	A0	2	2
Page zéro	LDY Oper	A4	2	3
Page zéro, X	LDY Oper, X	B4	2	4
Absolu	LDY Oper	AC	3	4
Absolu, X	LDY Oper, X	BC	3	4*

* Ajouter 1 si l'on passe la limite de page.

LSR — LSR *Décalage d'un bit vers la droite (mémoire ou accumulateur)* — LSR

Opération: 0 → | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | → C

N	Z	C	I	D	V
0	✓	✓	—	—	—

(Réf.: 10.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Accumulateur	LSR A	4A	1	2
Page zéro	LSR Oper	46	2	5
Page zéro, X	LSR Oper, X	56	2	6
Absolu	LSR Oper	4E	3	6
Absolu, X	LSR Oper, X	5E	3	7

NOP — NOP *Pas d'opération* — NOP

Opération: Pas d'opération (2 cycles)

N	Z	C	I	D	V
—	—	—	—	—	—

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	NOP	EA	1	2

ORA — ORA *Opération "OU" de la mémoire avec l'accumulateur* — ORA

Opération: A V M → A

N Z C I D V
✓ ✓ — — — —

(Réf.: 2.2.3.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	ORA #Oper	Ø9	2	2
Page zéro	ORA Oper	Ø5	2	3
Page zéro, X	ORA Oper, X	15	2	4
Absolu	ORA Oper	ØD	3	4
Absolu, X	ORA Oper, X	1D	3	4*
Absolu, Y	ORA Oper, Y	19	3	4*
(Indirect, X)	ORA (Oper, X)	Ø1	2	6
(Indirect), Y	ORA (Oper), Y	11	2	5

* Ajouter 1 si l'on passe une page.

PHA — PHA *Accumulateur placé sur la pile* — PHA

Opération: A ↓

N Z C I D V
— — — — — —

(Réf.: 8.5)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	PHA	48	1	3

PHP — PHP *État de l'unité de traitement placé sur la pile* — PHP

Opération: P↓

N Z C I D V
— — — — — —

(Réf.: 8.11)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	PHP	Ø8	1	3

PLA — PLA *Accumulateur extrait de la pile* — PLA

Opération: A ↑

N Z C I D V
✓ ✓ — — — —

(Réf.: 8.6)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	PLA	68	1	4

PLP — PLP *État de l'unité de traitement extrait de la pile* — PLP

Opération: A ↑

N Z C I D V
De la pile

(Réf.: 8.12)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	PLP	28	1	4

ROL — ROL *Rotation d'un bit vers la gauche (mémoire ou accumulateur)* — ROL

Opération: M ou A: 7 6 5 4 3 2 1 0 ← C ←

N Z C I D V
✓ ✓ — — — —

(Réf.: 10.3)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Accumulateur	ROL A	2A	1	2
Page zéro	ROL Oper	26	2	5
Page zéro, X	ROL Oper, X	36	2	6
Absolu	ROL Oper	2E	3	6
Absolu, X	ROL Oper, X	3E	3	7

ROR ROR *Rotation d'un bit vers la droite (mémoire ou accumulateur)* ROR

Opération: → C → 7 6 5 4 3 2 1 0 → (retour vers C)

N Z C I D V
✓ ✓ ✓ — — —

(Réf.: 10.4)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Accumulateur	ROR A	6A	1	2
Page zéro	ROR Oper	66	2	5
Page zéro, X	ROR Oper, X	76	2	6
Absolu	ROR Oper	6E	3	6
Absolu, X	ROR Oper, X	7E	3	7

Remarque: L'instruction ROR est disponible avec les microprocesseurs MCS650X depuis juin 1976.

RTI RTI *Retour d'une interruption* RTI

Opération: P↑ PC↑

N Z C I D V
De la pile

(Réf.: 9.6)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	RTI	40	1	6

RTS RTS *Retour d'un sous-programme* RTS

Opération: PC↑, PC + 1→PC

N Z C I D V
— — — — — —

(Réf.: 8.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	RTS	60	1	6

SBC — SBC *Soustraction de la mémoire de l'accumulateur avec retenue* — SBC

Opération: A − M − $\overline{C}$ → A

Remarque: $\overline{C}$ = Retenue

N	Z	C	I	D	V
✓	✓	✓	—	—	✓

(Réf.: 2.2.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Immédiat	SBC #Oper	E9	2	2
Page zéro	SBC Oper	E5	2	3
Page zéro, X	SBC Oper, X	F5	2	4
Absolu	SBC Oper	ED	3	4
Absolu, X	SBC Oper, X	FD	3	4*
Absolu, Y	SBC Oper, Y	F9	3	4*
(Indirect, X)	SBC (Oper, X)	E1	2	6
(Indirect), Y	SBC (Oper), Y	F1	2	5*

* Ajouter 1 si l'on passe la limite de page.

SEC — SEC *Établissement de l'indicateur de report* — SEC

Opération: 1 → C

N	Z	C	I	D	V
—	—	1	—	—	—

(Réf.: 3.0.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	SEC	38	1	2

SED — SED *Établissement du mode décimal* — SED

Opération: 1 → D

N	Z	C	I	D	V
—	—	—	—	1	—

(Réf.: 3.3.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	SED	F8	1	2

SEI — SEI *Établissement de l'état d'invalidation d'interruption* — SEI

Opération: 1 → I

N	Z	C	I	D	V
—	—	—	1	—	—

(Réf.: 3.2.1)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	SEI	78	1	2

STA — STA *Stockage de l'accumulateur en mémoire* — STA

Opération: A → M

N	Z	C	I	D	V
—	—	—	—	—	—

(Réf.: 2.1.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Page zéro	STA Oper	85	2	3
Page zéro, X	STA Oper, X	95	2	4
Absolu	STA Oper	8D	3	4
Absolu, X	STA Oper, X	9D	3	5
Absolu, Y	STA Oper, Y	99	3	5
(Indirect, X)	STA (Oper, X)	81	2	6
(Indirect), Y	STA (Oper), Y	91	2	6

STX — STX *Stockage de l'index X en mémoire* — STX

Opération: X → M

N	Z	C	I	D	V
—	—	—	—	—	—

(Réf.: 7.2)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Page zéro	STX Oper	86	2	3
Page zéro, Y	STX Oper, Y	96	2	4
Absolu	STX Oper	8E	3	4

STY

STY *Stockage de l'index Y en mémoire* **STY**

Opération: Y → M

N Z C I D V

— — — — — —

(Réf.: 7.3)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Page zéro	STY Oper	84	2	3
Page zéro, X	STY Oper, X	94	2	4
Absolu	STY Oper	8C	3	4

TAX

TAX *Transfert de l'accumulateur à l'index X* **TAX**

Opération: A → X

N Z C I D V

✓ ✓ — — — —

(Réf.: 7.11)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	TAX	AA	1	2

TAY

TAY *Transfert de l'accumulateur à l'index Y* **TAY**

Opération: A → Y

N Z C I D V

✓ ✓ — — — —

(Réf.: 7.13)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	TAY	A8	1	2

TSX

TSX *Transfert du pointeur de pile à l'index X*

TSX

Opération: S → X

N Z C I D V

✓ ✓ — — — —

(Réf.: 8.9)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	TSX	BA	1	2

TXA

TXA *Transfert de l'index X à l'accumulateur*

TXA

Opération: X → A

N Z C I D V

✓ ✓ — — — —

(Réf.: 7.12)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	TXA	8A	1	2

TXS

TXS *Transfert de l'index X au pointeur de pile*

TXS

Opération: X → S

N Z C I D V

— — — — — —

(Réf. 8.8)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	TXS	9A	1	2

TYA

TYA *Transfert de l'index Y à l'accumulateur*

TYA

Opération: Y → A

N Z C I D V

✓ ✓ — — — —

(Réf.: 7.14)

Mode d'adressage	Forme en langage d'assemblage	Code OP	Nbre d'octets	Nbre de cycles
Implicite	TYA	98	1	2

MODES D'ADRESSAGE DES INSTRUCTIONS ET

	Accumulateur	Immédiat	Page zéro	Page zéro, X	Page zéro, Y	Absolu	Absolu, X	Absolu, Y	Implicite	Relatif	(Indirect, X)	(Indirect), Y	Indirect absolu
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	.	6	7	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	.	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5

* Ajouter un cycle si l'indexation passe la limite de page

** Ajouter un cycle si le branchement est pris; ajouter un de plus

DURÉES D'EXÉCUTION CONNEXES (en cycles d'horloge)

	Accumulateur	Immédiat	Page zéro	Page zéro, X	Page zéro, Y	Absolu	Absolu, X	Absolu, Y	Implicite	Relatif	(Indirect, X)	(Indirect), Y	Indirect absolu
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX	.	.	3	.	4	4	.	.	.	.	.	.	.
STY	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

si l'opération de branchement passe la limite de page

00 — BRK
01 — ORA — (Indirect,X)
02 — Extension future
03 — Extension future
04 — Extension future
05 — ORA — Page zéro
06 — ASL — Page zéro
07 — Extension future
08 — PHP
09 — ORA — Immédiat
0A — ASL — Accumulateur
0B — Extension future
0C — Extension future
0D — ORA — Absolu
0E — ASL — Absolu
0F — Extension future
10 — BPL
11 — ORA — (Indirect), Y
12 — Extension future
13 — Extension future
14 — Extension future
15 — ORA — Page zéro,X
16 — ASL — Page zéro,X
17 — Extension future
18 — CLC
19 — ORA — Absolu,Y
1A — Extension future
1B — Extension future
1C — Extension future
1D — ORA — Absolu,X
1E — ASL — Absolu,X
1F — Extension future

20 —JSR
21 — AND — (Indirect,X)
22 — Extension future
23 — Extension future
24 — BIT — Page zéro
25 — AND — Page zéro
26 — ROL — Page zéro
27 — Extension future
28 — PLP
29 — AND — Immédiat
2A — ROL — Accumulateur
2B — Extension future
2C — BIT — Absolu
2D — AND — Absolu
2E — ROL — Absolu
2F — Extension future
30 — BMI
31 — AND — (Indirect), Y
32 — Extension future
33 — Extension future
34 — Extension future
35 — AND — Page zéro,X
36 — ROL — Page zéro,X
37 — Extension future
38 — SEC
39 — AND — Absolu,Y
3A — Extension future
3B — Extension future
3C — Extension future
3D — AND — Absolu,X
3E — ROL — Absolu,X
3F — Extension future

40 — RTI
41 — EOR — (Indirect,X)
42 — Extension future
43 — Extension future
44 — Extension future
45 — EOR — Page zéro
46 — LSR — Page zéro
47 — Extension future
48 — PHA
49 — EOR — Immédiat
4A — LSR — Accumulateur
4B — Extension future
4C — JMP—Absolu
4D — EOR — Absolu
4E — LSR — Absolu
4F — Extension future
50 — BVC
51 — EOR — (Indirect),Y
52 — Extension future
53 — Extension future
54 — Extension future
55 — EOR — Page zéro,X
56 — LSR — Page zéro,X
57 — Extension future
58 — CLI
59 — EOR — Absolu,Y
5A — Extension future
5B — Extension future
5C — Extension future
5D — EOR — Absolu,X
5E — LSR — Absolu,X
5F — Extension future

60 — RTS
61 — ADC — (Indirect,X)
62 — Extension future
63 — Extension future
64 — Extension future
65 — ADC — Page zéro
66 — ROR — Page zéro
67 — Extension future
68 — PLA
69 — ADC — Immédiat
6A — ROR — Accumulateur
6B — Extension future
6C — JMP — Indirect
6D — ADC — Absolu
6E — ROR — Absolu
6F — Extension future
70 — BVS
71 — ADC — (Indirect),Y
72 — Extension future
73 — Extension future
74 — Extension future
75 — ADC — Page zéro,X
76 — ROR — Page zéro,X
77 — Extension future
78 — SEI
79 — ADC — Absolu,Y
7A — Extension future
7B — Extension future
7C — Extension future
7D — ADC — Absolu,X
7E — ROR — Absolu,X
7F — Extension future

80 — Extension future
81 — STA — (Indirect,X)
82 — Extension future
83 — Extension future
84 — STY — Page zéro
85 — STA — Page zéro
86 — STX — Page zéro
87 — Extension future
88 — DEY
89 — Extension future
8A — TXA
8B — Extension future
8C — STY—Absolu
8D — STA — Absolu
8E — STX — Absolu
8F — Extension future
90 — BCC
91 — STA — (Indirect),Y
92 — Extension future
93 — Extension future
94 — STY — Page zéro,X
95 — STA — Page zéro,X
96 — STX — Page zéro,Y
97 — Extension future
98 — TYA
99 — STA — Absolu,Y
9A — TXS
9B — Extension future
9C — Extension future
9D — STA — Absolu,X
9E — Extension future
9F — Extension future

A0 — LDY — Immédiat
A1 — LDA — (Indirect,X)
A2 — LDX — Immédiat
A3 — Extension future
A4 — LDY — Page zéro
A5 — LDA — Page zéro
A6 — LDX — Page zéro
A7 — Extension future
A8 — TAY
A9 — LDA — Immédiat
AA — TAX
AB — Extension future
AC — LDY — Absolu
AD — LDA — Absolu
AE — LDX — Absolu
AF — Extension future
B0 — BCS
B1 — LDA — (Indirect),Y
B2 — Extension future
B3 — Extension future
B4 — LDY — Page zéro,X
B5 — LDA — Page zéro,X
B6 — LDX — Page zéro,Y
B7 — Extension future
B8 — CLV
B9 — LDA — Absolu,Y
BA — TSX
BB — Extension future
BC — LDY — Absolu X
BD — LDA — Absolu,X
BE — LDX — Absolu,Y
BF — Extension future

CØ — CPY — Immédiat
C1 — CMP — (Indirect,X)
C2 — Extension future
C3 — Extension future
C4 — CPY — Page zéro
C5 — CMP — Page zéro
C6 — DEC — Page zéro
C7 — Extension future
C8 — INY
C9 — CMP — Immédiat
CA — DEX
CB — Extension future
CC — CPY — Absolu
CD — CMP — Absolu
CE — DEC — Absolu
CF — Extension future
DØ — BNE
D1 — CMP — (Indirect),Y
D2 — Extension future
D3 — Extension future
D4 — Extension future
D5 — CMP — Page zéro,X
D6 — DEC — Page zéro,X
D7 — Extension future
D8 — CLD
D9 — CMP — Absolu,Y
DA — Extension future
DB — Extension future
DC — Extension future
DD — CMP — Absolu,X
DE — DEC — Absolu,X
DF — Extension future

EØ — CPX — Immédiat
E1 — SBC — (Indirect,X)
E2 — Extension future
E3 — Extension future
E4 — CPX — Page zéro
E5 — SBC — Page zéro
E6 — INC — Page zéro
E7 — Extension future
E8 — INX
E9 — SBC — Immédiat
EA — NOP
EB — Extension future
EC — CPX — Absolu
ED — SBC — Absolu
EE — INC — Absolu
EF — Extension future
FØ — BEQ
F1 — SBC — (Indirect),Y
F2 — Extension future
F3 — Extension future
F4 — Extension future
F5 — SBC — Page zéro,X
F6 — INC — Page zéro,X
F7 — Extension future
F8 — SED
F9 — SBC — Absolu,Y
FA — Extension future
FB — Extension future
FC — Extension future
FD — SBC — Absolu,X
FE — INC — Absolu,Y
FF — Extension future

GESTION DE MÉMOIRE AVEC LE COMMODORE 64

Le Commodore 64 a une mémoire vive (RAM) de 64 K-octets. Sa mémoire morte (ROM) de 20 K-octets contient le BASIC, le système d'exploitation et le jeu de caractères normaux. Il accède aussi aux dispositifs d'entrée/sortie qui occupent 4 K de mémoire. Comment parvient-on à une telle diversité avec un bus d'adresses de 16 bits, normalement capable d'accéder à 64 K?

Le secret réside dans la microplaquette 6510 qui est équipée d'un accès d'entrée/sortie. Cet accès sert à contrôler si la mémoire vive, la mémoire morte ou l'entrée/sortie apparaît dans certaines parties de la mémoire du système. L'accès sert aussi à commander le magnétocassette Datasette TM; il importe donc d'affecter les bits corrects.

L'accès d'entrée/sortie de la 6510 apparaît à la position 1. Le registre de direction de données de cet accès apparaît à la position 0. Le contrôle de cet accès est analogue à celui des autres accès d'entrée/sortie du système; la direction des données contrôle si un bit donné est une entrée ou une sortie; le transfert réel des données se produit dans l'accès lui-même.

Les lignes de l'accès de commande 6510 se définissent ainsi:

NOM	BIT	DIRECTION	DESCRIPTION
LORAM	0	SORTIE	Commande pour RAM/ROM à $A000—$BFFF (BASIC)
HIRAM	1	SORTIE	Commande pour RAM/ROM à $E000—$FFFF (KERNAL)
CHAREN	2	SORTIE	Commande pour entrée/sortie/ROM à $D000— $DFFF
	3	SORTIE	Ligne d'écriture de cassette
	4	ENTRÉE	Détection de sélecteur de cassette
	5	SORTIE	Commande de moteur de cassette

La valeur correcte de registre de direction de données est:

BITS	5	4	3	2	1	0
	1	0	1	1	1	1

(dans laquelle 1 est une sortie et 0 une entrée).

On obtient ainsi une valeur de 47 en décimal. Le Commodore 64 établit automatiquement le registre de direction de données à cette valeur.

En général, les lignes de commande exécutent la fonction indiquée dans leur description. Il arrive cependant qu'on utilise une combinaison de lignes de commande pour arriver à une configuration particulière de mémoire.

On peut généralement imaginer **LORAM** (bit 0—RAM basse) comme une ligne de commande qui introduit la mémoire morte de BASIC de 8 K-octets dans l'espace d'adresses du microprocesseur et l'en extrait. Cette ligne est normalement à l'état haut pour l'exploitation en BASIC. Si cette ligne est programmée à l'état bas, la mémoire morte de BASIC disparaît de la topographie de mémoire pour être remplacée par les 8 K-octets de mémoire vive de $A000 à $BFFF.

On peut généralement imaginer **HIRAM** (bit 1—RAM haute), comme une ligne de commande qui introduit la mémoire morte de KERNAL de 8 K-octets dans l'espace d'adresses du microprocesseur et l'en extrait. Cette ligne est normalement à l'état haut pour l'exploitation en BASIC. Si cette ligne est programmée à l'état bas, la mémoire morte de KERNAL disparaît de la topographie de mémoire pour être remplacée par les 8 K-octets de mémoire vive de $E000 à $FFFF.

CHAREN (bit 2) ne s'utilise que pour introduire la mémoire morte de générateur de caractère de 4 K-octets dans l'espace d'adresses du microprocesseur ou l'en extraire. Sur le plan de l'unité de traitement, la mémoire morte de caractères occupe le même espace d'adresses que les dispositifs d'entrée/sortie ($D000—$DFFF). Si la ligne CHAREN est fixée à 1 (état normal), les dispositifs d'entrée/sortie apparaissent dans l'espace d'adresses du microprocesseur et on ne peut pas accéder à la mémoire morte de caractères. Quand le bit CHAREN est mis à 0, la mémoire morte de caractères apparaît dans l'espace d'adresses d'unité de traitement; on ne peut plus accéder aux dispositifs d'entrée/sortie. (Le microprocesseur ne doit accéder à la mémoire morte de caractères que pendant le téléchargement du jeu de caractères de la mémoire morte à la mémoire vive. Cette opération demande une attention particulière; voir la section sur les caractères programmables, dans le chapitre sur les graphiques.) On peut substituer la ligne CHAREN par d'autres lignes de commande dans certaines configurations de mémoire. CHAREN n'a aucun effet sur une configuration de mémoire sans dispositifs d'entrée/sortie. La mémoire vive apparaît à la place de $D000 à $DFFF.

REMARQUE: Dans une topographie de mémoire contenant la mémoire morte, une instruction d'écriture (POKE) dans une position de mémoire morte stocke les données en mémoire vive (RAM), "sous" mémoire ROM. L'écriture dans une position de mémoire morte stocke les données dans la mémoire vive "cachée". On peut ainsi tenir un écran à haute définition "sous" une mémoire morte et le changer sans devoir remettre l'écran dans l'espace d'adresses de l'unité de traitement. Une lecture (READ) de position de mémoire morte retourne naturellement le contenu de la mémoire morte et non la mémoire vive "cachée".

TOPOGRAPHIE DE MÉMOIRE FONDAMENTALE DU COMMODORE 64

Adresses	Contenu
E000-FFFF	MÉMOIRE RAM OU ROM DE KERNAL 8 K
D000-DFFF	ENTRÉE/SORTIE OU MÉMOIRE RAM 4 K OU MÉMOIRE ROM DE CARACTÈRES
C000-CFFF	MÉMOIRE RAM 4 K
A000-BFFF	MÉMOIRE ROM BASIC 8 K, RAM OU ROM ENFICHABLE
8000-9FFF	MÉMOIRE RAM 8 K OU ROM ENFICHABLE
4000-7FFF	MÉMOIRE RAM 16 K
0000-3FFF	MÉMOIRE RAM 16 K

DÉTAILS D'ENTRÉE/SORTIE

D000-D3FF	**VIC (Contrôleur vidéo)**	**1 K-octet**
D400-D7FF	**SID (Synthétiseur de son)**	**1 K-octet**
D800-DBFF	**Mémoire vive de couleurs**	**1 K-octet/2**
DC00-DCFF	**CIA1 (Clavier)**	**256 octets**
DD00-DDFF	**CIA2 (Bus série, accès utilisateur/RS-232)**	**256 octets**
DE00-DEFF	**Fente d'entrée/sortie ouverte n° 1 (validation CP/M)**	**256 octets**
DF00-DFFF	**Fente d'entrée/sortie ouverte n° 2 (disque)**	**256 octets**

Les deux fentes d'entrée/sortie ouvertes servent à l'entrée/sortie générale d'utilisateur, aux cartouches d'entrée/sortie spéciales (IEEE par exemple). Il se peut qu'elles servent à la validation de la cartouche Z-80 (option CP/M) et pour l'interface avec un système économique à disques à grande vitesse.

Le système permet le départ automatique du programme dans une cartouche d'extension de Commodore 64. Le programme sur cartouche démarre si les neuf premiers octets de la mémoire morte de cartouche, à partir de la position 32768 ($8000) contiennent des données spécifiques. Les deux premiers octets doivent contenir le vecteur de départ à froid utilisé par le programme sur cartouche. Les deux octets suivants, à la position 32770 ($8002), doivent correspondre au vecteur de départ à chaud utilisé par le programme sur cartouche. Les trois octets suivants doivent correspondre aux lettres CBM, avec le bit 7 fixé dans chaque lettre. Les deux derniers octets doivent correspondre aux chiffres "80" en ASCII de PET.

TOPOGRAPHIES DE MÉMOIRE DU COMMODORE 64

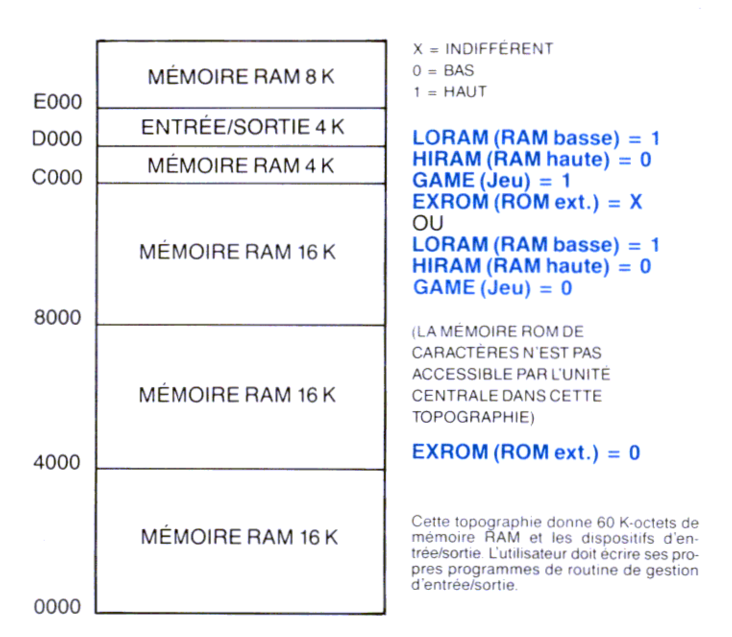
Les tables suivantes donnent les diverses configurations de mémoire permises avec le Commodore 64, les états des lignes de commande qui choisissent chaque topographie de mémoire et l'usage prévu de chaque topograhie.

X = INDIFFÉRENT
0 = BAS
1 = HAUT

Adresse	Contenu
	MÉMOIRE ROM DE KERNAL 8 K
E000	ENTRÉE/SORTIE 4 K
D000	MÉMOIRE RAM 4 K (TAMPON)
C000	MÉMOIRE ROM DE BASIC 8 K
A000	MÉMOIRE RAM 8 K
8000	MÉMOIRE RAM 16 K
4000	MÉMOIRE RAM 16 K
0000	

LORAM (RAM basse) = 1
HIRAM (RAM haute) = 1
GAME (jeu) = 1
EXROM (ROM ext.) = 1

Topographie de mémoire BASIC implicite qui donne le BASIC 2.0 et 38 K-octets contigus de mémoire RAM d'utilisateur.



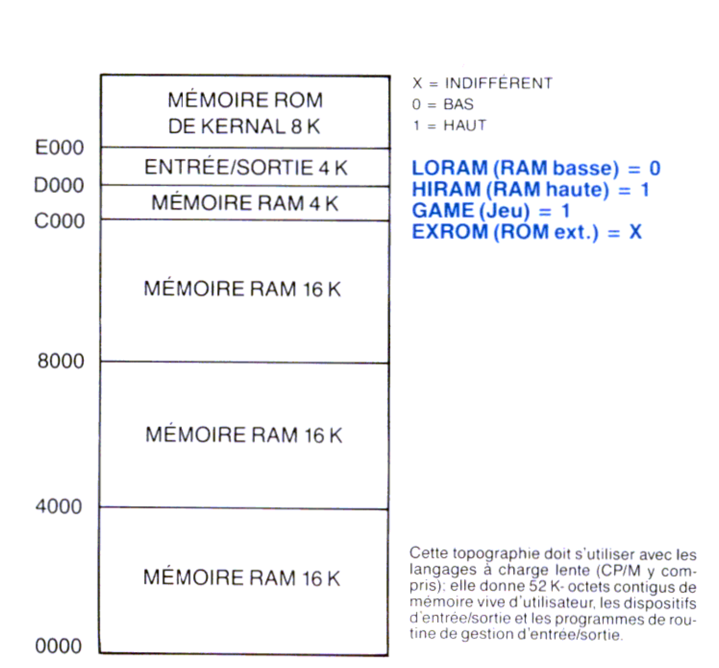
X = INDIFFÉRENT
0 = BAS
1 = HAUT

LORAM (RAM basse) = 1
HIRAM (RAM haute) = 0
GAME (Jeu) = 1
EXROM (ROM ext.) = X
OU
LORAM (RAM basse) = 1
HIRAM (RAM haute) = 0
GAME (Jeu) = 0

(LA MÉMOIRE ROM DE CARACTÈRES N'EST PAS ACCESSIBLE PAR L'UNITÉ CENTRALE DANS CETTE TOPOGRAPHIE)

EXROM (ROM ext.) = 0

Cette topographie donne 60 K-octets de mémoire RAM et les dispositifs d'entrée/sortie. L'utilisateur doit écrire ses propres programmes de routine de gestion d'entrée/sortie.



X = INDIFFÉRENT
0 = BAS
1 = HAUT

LORAM (RAM basse) = 0
HIRAM (RAM haute) = 1
GAME (Jeu) = 1
EXROM (ROM ext.) = X

Cette topographie doit s'utiliser avec les langages à charge lente (CP/M y compris); elle donne 52 K-octets contigus de mémoire vive d'utilisateur, les dispositifs d'entrée/sortie et les programmes de routine de gestion d'entrée/sortie.

	MÉMOIRE RAM 16 K
C000	MÉMOIRE RAM 16 K
8000	MÉMOIRE RAM 16 K
4000	MÉMOIRE RAM 16 K
0000	

X = INDIFFÉRENT
0 = BAS
1 = HAUT

LORAM (RAM basse) = 0
HIRAM (RAM haute) = 0
GAME (Jeu) = 1
EXROM (ROM ext.) = X
OU
LORAM (RAM basse) = 0
HIRAM (RAM haute) = 0
GAME (Jeu) = X
EXROM (ROM ext.) = 0

Cette topographie permet d'accéder aux 64 K-octets de mémoire RAM. Les dispositifs d'entrée/sortie doivent être ramenés dans l'espace d'adresses de l'unité de traitement pour toute opération d'entrée/sortie.

	MÉMOIRE ROM DE KERNAL 8 K
E000	ENTRÉE/SORTIE 4 K
D000	MÉMOIRE RAM 4 K (TAMPON)
C000	MÉMOIRE ROM BASIC 8 K
A000	CARTOUCHE DE MÉMOIRE ROM 8 K (EXT. BASIC)
8000	MÉMOIRE RAM 16 K
4000	MÉMOIRE RAM 16 K
0000	

X = INDIFFÉRENT
0 = BAS
1 = HAUT

LORAM (RAM basse) = 1
HIRAM (RAM haute) = 1
GAME (Jeu) = 0
EXROM (ROM ext.) = 0

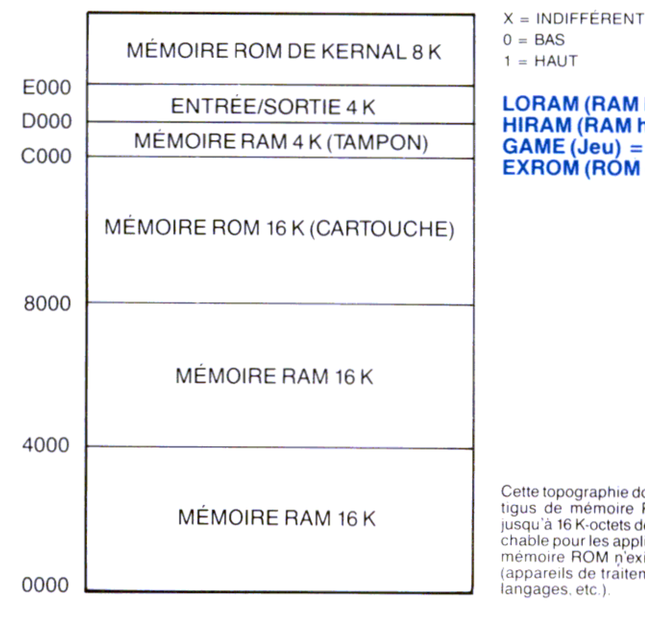
Configuration standard pour un système BASIC avec mémoire ROM d'extension BASIC. Cette topographie donne 32 K-octets contigus de mémoire RAM d'utilisateur et jusqu'à 8 K-octets de "perfectionnement" BASIC.

MÉMOIRE ROM DE KERNAL 8 K
E000
ENTRÉE/SORTIE 4 K
D000
MÉMOIRE RAM 4 K (TAMPON)
C000
MÉMOIRE ROM 8 K (CARTOUCHE)
A000
MÉMOIRE RAM 8 K
8000
MÉMOIRE RAM 16 K
4000
MÉMOIRE RAM 16 K
0000

X = INDIFFÉRENT
0 = BAS
1 = HAUT

LORAM (RAM basse) = 0
HIRAM (RAM haute) = 1
GAME (Jeu) = 0
EXROM (ROM ext.) = 0

Cette topographie donne 40 K-octets de mémoire RAM d'utilisateur et jusqu'à 8 K-octets de mémoire ROM enfichable pour les applications spéciales en mémoire ROM n'exigeant pas le BASIC.



X = INDIFFÉRENT
0 = BAS
1 = HAUT

LORAM (RAM basse) = 1
HIRAM (RAM haute) = 1
GAME (Jeu) = 0
EXROM (ROM ext.) = 0

Cette topographie donne 32 K-octets contigus de mémoire RAM d'utilisateur et jusqu'à 16 K-octets de mémoire ROM enfichable pour les applications spéciales en mémoire ROM n'exigeant pas le BASIC (appareils de traitement de texte, autres langages, etc.).

Adresse	
	MÉMOIRE ROM 8 K (CARTOUCHE)
E000	ENTRÉE/SORTIE 4 K
D000	OUVERTE 4 K
C000	OUVERTE 8 K
A000	MÉMOIRE ROM 8 K (CARTOUCHE)
8000	OUVERTE 16 K
4000	OUVERTE 12 K
1000	MÉMOIRE RAM 4 K
0000	

X = INDIFFÉRENT
0 = BAS
1 = HAUT

LORAM (RAM basse) = X
HIRAM (RAM haute) = X
GAME (Jeu) = 0
EXROM (ROM ext.) = 1

Topographie de mémoire du jeu vidéo ULTIMAX. On peut remarquer que l'on accède à la mémoire RAM d'extension de 2 K-octets de l'ULTIMAX par le COMMODORE 64; toute mémoire RAM dans la cartouche est ignorée.

LE KERNAL

Dans le domaine des micro-ordinateurs, les programmeurs doivent faire face aux changements apportés au système d'exploitation d'un ordinateur par son fabricant. Les programmes en langage machine, dont la mise au point a demandé beaucoup de temps, peuvent ne plus fonctionner et obliger le programmeur à leur intégrer des révisions importantes. Pour remédier à ce problème, Commodore a mis au point une méthode de protection du programmeur dite "**KERNAL**".

Le KERNAL est essentiellement une **TABLE DE SAUTS** normalisée pour les programmes de routine d'entrée, de sortie et de gestion de mémoire dans le système d'exploitation. Les positions de chaque programme de routine en mémoire morte peuvent changer avec l'amélioration du système, mais la table de sauts KERNAL est toujours adaptée aux changements. Si les programmes en langage machine n'utilisent que les sous-programmes en mémoire morte de système par l'intermédiaire du KERNAL, les modifications prennent alors beaucoup moins de temps, s'il arrive qu'elles soient nécessaires.

Le KERNAL constitue le système d'exploitation de l'ordinateur Commodore 64. Il commande l'entrée, la sortie et la gestion de mémoire.

Pour simplifier les programmes en langage machine que l'on écrit et pour s'assurer que les versions ultérieures du système d'exploitation du Commodore 64 ne rendent pas inutiles les programmes déjà écrits en langage machine, le KERNAL possède une table de sauts. En profitant des 39 programmes de routine d'entrée/sortie et d'autres programmes utilitaires indiqués par la table, on peut économiser du temps, mais on facilite aussi le transfert des programmes d'un ordinateur Commodore à un autre.

La table de sauts se trouve en dernière page, dans la mémoire morte (ROM).

Pour utiliser la table de sauts KERNAL, il faut d'abord fixer les paramètres nécessaires au programme de routine KERNAL. Sauter ensuite au programme de routine **(JSR)** à l'endroit approprié de la table de sauts KERNAL. Après avoir rempli sa fonction, le KERNAL rend la commande au programme en langage machine. Suivant le sous-programme de KERNAL utilisé, certains registres peuvent ramener les paramètres au programme. On peut trouver les registres particuliers de chaque programme de routine KERNAL dans les descriptions individuelles de ces programmes de routine.

À ce point, on peut s'interroger sur l'intérêt de la table de sauts. Pourquoi ne pas sauter directement au programme de routine (JSR) KERNAL en cause? En cas de changement du KERNAL ou du BASIC, la table de sauts permet de continuer à utiliser les programmes en langage machine. Dans les futurs systèmes d'exploitation, les positions de mémoire des programmes de routine seront déplacées à des positions différentes dans la topographie de mémoire, mais la table de sauts continuera à fonctionner correctement.

OPÉRATIONS DE MISE SOUS TENSION DU KERNAL

1) À la mise sous tension, le KERNAL remet d'abord le pointeur de pile à l'état initial et efface le mode décimal.

2) Le KERNAL vérifie ensuite la présence d'une cartouche de mémoire morte de mise en marche automatique à la position $8000 HEX (32768 en décimal). Si elle est présente, l'initialisation normale est suspendue et la commande est transférée au code de cartouche. S'il n'y a pas de mémoire morte de mise en marche automatique, l'initialisation normale du système continue.

3) Le KERNAL initialise ensuite toutes les dispositions d'entrée/sortie. Le bus série est initialisé. Les deux microplaquettes CIA 6526 sont fixées aux valeurs correctes pour l'exploration du clavier. La minuterie de 60 Hz est mise en fonction. La microplaquette SID est effacée. La topographie de mémoire de BASIC est choisie et le moteur du magnétocassette est coupé.

4) Le KERNAL exécute ensuite une vérification de mémoire vive en fixant les pointeurs supérieur et inférieur de mémoire. En outre, la page zéro est initialisée et le tampon de cassette est établi.

Le sous-programme de vérification de mémoire RAM est une vérification non destructive qui commence à la position $0300 et continue dans le sens ascendant. Quand la vérification a localisé la première position qui n'appartient pas à la mémoire vive, le pointeur du haut de la mémoire vive RAM est fixé. Le bas de la mémoire est toujours établi à $0800; la disposition d'écran est toujours à $0400.

5) Pour terminer, le KERNAL exécute les autres activités suivantes. Les vecteurs d'entrée/sortie sont fixés à des valeurs implicites. La table de sauts indirects en mémoire basse est établie. L'écran est ensuite effacé et toutes les variables d'éditeur d'écran sont remises à l'état initial. L'adresse indirecte à $A000 sert à la mise en fonction du BASIC.

UTILISATION DU KERNAL

Quand on écrit des programmes en langage machine, il est souvent pratique d'utiliser des programmes de routine déjà dans le système d'exploitation pour l'entrée/sortie, l'accès à l'horloge du système, la gestion de la mémoire et les opérations similaires. Il est absolument inutile d'écrire des programmes de routine à plusieurs reprises, car la facilité d'accès au système d'exploitation contribue à accélérer la programmation en langage machine.

Nous avons déjà vu que le KERNAL est une table de sauts qui n'est qu'une réunion d'instructions JMP à de nombreux programmes de routine du système d'exploitation.

Pour utiliser un programme de routine KERNAL, on doit d'abord procéder à toutes les préparations exigées par ce programme de routine. Si un programme de routine indique que l'on doit d'abord appeler un autre programme de routine KERNAL, il faut alors procéder à cette opération. Si le programme de routine indique de placer un nombre dans l'accumulateur, il faut également le faire. Si l'on ne respecte pas ces conditions, les programmes de routine ne risquent guère de donner les résultats escomptés.

Quand les préparations sont terminées, appeler le programme de routine à l'aide de l'instruction JSR. Tous les programmes de routine accessibles de KERNAL sont structurés sous forme de sous-programmes qu'il faut terminer par une instruction RTS. Quand le programme de routine KERNAL a accompli sa tâche, la commande est renvoyée au programme, à l'instruction qui suit JSR.

De nombreux programmes de routine KERNAL retournent des codes d'erreur dans le mot d'état ou l'accumulateur en cas de problème dans leur déroulement. Les bonnes habitudes de programmation et la réussite des programmes en langage machine exigent de traiter convenablement ces erreurs. Si l'on ignore un retour d'erreur, le reste du programme risque de conduire à un échec.

Quand on utilise le KERNAL, il suffit de se tenir aux trois étapes suivantes:

1) Établissement
2) Appel du programme de routine
3) Traitement des erreurs

On utilise les conventions suivantes dans la description des programmes de routine KERNAL:

—**NOM DE FONCTION:** Nom du programme de routine KERNAL.

—**ADRESSE D'APPEL:** Adresse d'appel du programme de routine KERNAL, en mode hexadécimal.

—**REGISTRES DE COMMUNICATION:** Les registres de ce groupe servent à la communication des paramètres avec les programmes de routine KERNAL.

—**PROGRAMMES DE PRÉPARATION:** Certains programmes KERNAL ne peuvent fonctionner qu'après l'établissement des données. Les programmes de routine nécessaires sont indiqués ici.

—**RETOURS D'ERREUR:** Un retour de programme de routine KERNAL, avec le report, indique qu'il s'est produit une erreur dans le traitement. L'accumulateur contient le numéro de l'erreur.

—**CONDITIONS DE PILE:** Nombre réel d'octets de pile utilisés par le programme de routine KERNAL.

—**REGISTRES AFFECTÉS:** Tous les registres utilisés par le programme de routine KERNAL sont indiqués ici.

—**DESCRIPTION:** On donne ici une courte explication de la fonction du programme KERNAL.

Nous donnons ci-après la liste des programmes de routine KERNAL.

PROGRAMMES DE ROUTINE KERNAL POUVANT ÊTRE APPELÉS PAR L'UTILISATEUR

NOM	ADRESSE		FONCTION
	HEX.	DÉCIMAL	
ACPTR	$FFA5	65445	Octet d'entrée de l'accès série.
CHKIN	$FFC6	65478	Ouverture d'un canal pour l'entrée
CHKOUT	$FFC9	65481	Ouverture d'un canal pour la sortie
CHRIN	$FFCF	65487	Caractère d'entrée venant du canal
CHROUT	$FFD2	65490	Caractère de sortie vers canal
CIOUT	$FFA8	65448	Octet de sortie vers accès série
CINT	$FF81	65409	Initialisation d'éditeur d'écran
CLALL	$FFE7	65511	Fermeture de tous les canaux et fichiers
CLOSE	$FFC3	65475	Fermeture d'un fichier logique spécifié
CLRCHN	$FFCC	65484	Fermeture des canaux d'entrée et de sortie
GETIN	$FFE4	65508	Extraction de caractère de la file d'attente de clavier (tampon de clavier)
IOBASE	$FFF3	65523	Retour de l'adresse de base des dispositifs d'entrée/sortie
IOINIT	$FF84	65412	Initialisation d'entrée/sortie
LISTEN	$FFB1	65457	Ordre d'écoute aux dispositifs sur le bus série
LOAD	$FFD5	65493	Chargement de la mémoire vive depuis un dispositif
MEMBOT	$FF9C	65436	Lecture/établissement du bas de la mémoire
MEMTOP	$FF99	65433	Lecture/établissement du haut de la mémoire
OPEN	$FFC0	65472	Ouverture d'un fichier logique
PLOT	$FFF0	65520	Lecture/établissement de la position de curseur X,Y

NOM	ADRESSE		FONCTION
	HEX.	DÉCIMAL	
RAMTAS	$FF87	65415	Initialisation de la mémoire vive, attribution du tampon de cassette et établissement de l'écran à $0400
RDTIM	$FFDE	65502	Lecture de l'horloge à temps réel
READST	$FFB7	65463	Lecture du mot d'état d'entrée/sortie
RESTOR	$FF8A	65418	Rétablissement des vecteurs d'entrée/sortie implicites
SAVE	$FFD8	65496	Sauvegarde de la mémoire vive sur dispositif
SCNKEY	$FF9F	65439	Exploration du clavier
SCREEN	$FFED	65517	Retour de l'organisation X,Y de l'écran
SECOND	$FF93	65427	Envoi de l'adresse secondaire après LISTEN
SETLFS	$FFBA	65466	Établissement des première et deuxième adresses logiques
SETMSG	$FF90	65424	Commande des messages de KERNAL
SETNAM	$FFBD	65469	Établissement du nom de fichier
SETTIM	$FFDB	65499	Établissement de l'horloge à temps réel
SETTMO	$FFA2	65442	Fixe l'heure d'arrêt au bus série
STOP	$FFE1	65505	Exploration de la touche d'arrêt
TALK	$FFB4	65460	Ordre d'émission de dispositif de bus série
TKSA	$FF96	65430	Envoi à l'adresse secondaire après émission
UDTIM	$FFEA	65514	Progression de l'horloge à temps réel
UNLSN	$FFAE	65454	Ordre de désadressage au bus série
UNTLK	$FFAB	65451	Ordre de coupure d'émission au bus série
VECTOR	$FF8D	65421	Lecture/établissement d'entrée/sortie vectorisée

B-1. Nom de fonction: ACPTR

But: Extraction de données du bus série
Adresse d'appel: $FFA5 (hexadécimal), 65445 (décimal)
Registres de communication: .A
Programmes de routine de préparation: TALK, TKSA
Retours d'erreur: Voir READST
Conditions de pile: 13
Registres affectés: .A, .X

Description: Ce programme de routine s'utilise pour extraire des informations d'un dispositif sur le bus série (disque par exemple). Ce programme extrait un octet de données du bus série par l'établissement d'une liaison intégrale. Les données sont retournées dans l'accumulateur. Pour se préparer à ce programme, on doit d'abord appeler le programme de routine TALK pour ordonner au dispositif sur le bus série d'envoyer des données par ce dernier. Si le dispositif d'entrée doit recevoir une commande secondaire, on doit l'envoyer à l'aide du programme TKSA de KERNAL avant d'appeler le programme de routine. Les erreurs sont retournées dans le mot d'état. Le programme de routine READST sert à lire le mot d'état.

Utilisation:

0) Ordonner à un dispositif sur le bus série de se préparer à envoyer des données au Commodore 64 (Utiliser les programmes de routine TALK et TKSA.)
1) Appeler ce programme de routine (avec JSR).
2) Stocker les données ou les utiliser.

EXEMPLE:

```
;EXTRACTION D'UN OCTET DU BUS
JSR ACPTR
STA DATA
```

B-2. Nom de fonction: CHKIN

But: Ouverture d'un canal pour l'entrée.
Adresse d'appel: $FFC6 (hexadécimal), 65478 (décimal)
Registres de communication: .X
Programmes de routine de préparation: (OPEN)
Retours d'erreur:
Conditions de pile: Aucune
Registres affectés: .A, .X

Description: On peut définir tout fichier logique déjà ouvert par le programme de routine KERNAL OPEN comme canal d'entrée. Le dispositif sur le canal doit évidemment être un dispositif d'entrée, sinon il se produit une erreur et le programme s'arrête.

Si l'on obtient des données d'un point autre que le clavier, appeler ce programme en utilisant les programmes CHRIN ou GETIN de KERNAL pour l'entrée des données. Si l'on veut utiliser l'entrée du clavier et si aucun autre canal d'entrée n'est ouvert, on n'a alors pas besoin des appels à ce programme de routine ou au programme OPEN.

Si l'on utilise ce programme de routine avec un dispositif sur le bus série, il envoie automatiquement l'adresse d'émission (et l'adresse secondaire s'il en a été spécifié une par le programme OPEN) par le bus.

Utilisation:

0) Ouvrir le fichier logique (le cas échéant; voir description ci-dessus).
1) Charger le registre .X avec le numéro du fichier logique à utiliser.
2) Appeler ce programme (avec une commande JSR).

Erreurs possibles:

#3: Fichier non ouvert
#5: Dispositif absent
#6: Le fichier n'est pas un fichier d'entrée

EXEMPLE:

```
; PRÉPARATION POUR L'ENTRÉE À PARTIR DU FICHIER LOGIQUE 2
LDX #2
JSR CHKIN
```

B-3. Nom de fonction: CHKOUT

But: Ouverture d'un canal pour la sortie
Adresse d'appel: $FFC9 (hexadécimal), 65481 (décimal)
Registres de communication: .X
Programmes de routine de préparation: (OPEN)
Retours d'erreur: 0,3,5,7 (Voir READST)
Conditions de pile: 4+
Registres affectés: .A, .X

Description: On peut définir comme canal de sortie tout numéro de fichier logique créé par le programme de routine KERNAL OPEN. Le dispositif sur lequel on compte ouvrir un canal doit évidemment être prévu pour la sortie, sinon il se produit une erreur et le programme s'interrompt.

On doit appeler ce programme avant d'envoyer des données à un dispositif de sortie, sauf si l'on désire utiliser l'écran du Commodore 64 comme dispositif de sortie. Si l'on désire la sortie vers l'écran et si aucun autre canal de sortie n'est déjà défini, les appels à ce programme de routine, ainsi qu'au programme de routine OPEN, sont inutiles.

Si l'on utilise ce programme de routine pour ouvrir un canal vers un dispositif sur le bus série, il envoie automatiquement l'adresse LISTEN spécifiée par le programme de routine OPEN (et une adresse secondaire, le cas échéant).

Utilisation:

RAPPEL: Ce programme N'EST PAS utile pour envoyer des données à l'écran.

0) Utiliser le programme de routine KERNAL OPEN pour spécifier un numéro de fichier logique, une adresse LISTEN et une adresse secondaire, le cas échéant.
1) Charger le registre .X avec le numéro de fichier logique utilisé dans l'instruction d'ouverture.
2) Appeler ce programme de routine (avec l'instruction JSR).

EXEMPLE:

```
LDX #3          ;DÉFINIT LE FICHIER LOGIQUE 3 COMME CANAL DE SORTIE
JSR CHKOUT
```

Erreurs possibles:

#3: Fichier non ouvert
#5: Dispositif absent
#7: Le fichier n'est pas un fichier d'entrée

B-4. Nom de fonction: CHRIN

But: Extraction d'un caractère du canal d'entrée
Adresse d'appel: $FFCF (hexadécimal), 65487 (décimal)
Registres de communication: .A
Programmes de routine de préparation: (OPEN, CHKIN)
Retours d'erreur: 0 (Voir READST)
Conditions de pile: 7+
Registres affectés: .A, .X

Description: Ce programme extrait un octet de données d'un canal déjà établi comme canal d'entrée par le programme de routine KERNAL CHKIN. Si l'on n'a pas utilisé CHKIN pour définir un autre canal d'entrée, toutes les autres données doivent alors venir du clavier. L'octet de données est retourné dans l'accumulateur. Le canal reste ouvert après l'appel.

L'entrée depuis le clavier suit un cheminement spécial. Tout d'abord, le curseur est mis en fonction et clignote jusqu'à ce qu'on tape un retour du chariot au clavier. Tous les caractères de la ligne (maximum de 88) sont stockés dans le tampon d'entrée de BASIC. On peut récupérer ces caractères les uns après les autres en appelant une fois ce programme pour chaque caractère. À la récupération du retour du chariot, la ligne complète a été traitée. À l'appel suivant de ce programme, le processus entier recommence . . . le curseur clignote, etc.

Utilisation:

À PARTIR DU CLAVIER

1) Extraire un octet de données en appelant ce programme de routine.
2) Stocker l'octet de données.
3) Vérifier s'il s'agit du dernier octet de données (CR?).
4) Si ce n'est pas le cas, aller à l'étape 1.

EXEMPLE:

```
        LDY $#00      ;PRÉPARE LE REGISTRE .Y POUR STOCKER DES
                      ;DONNÉES
RD      JSR CHRIN
        STA DATA,Y    ;STOCKE LE Ye OCTET DE DONNÉES DANS LA
                      ;Ye POSITION DE LA ZONE DE DONNÉES
        INY
        CMP #CR       ;S'AGIT-IL D'UN RETOUR DU CHARIOT?
        BNE RD        ;NON; EXTRAIRE UN AUTRE OCTET DE DONNÉES
```

EXEMPLE:

```
JSR CHRIN
STA DATA
```

À PARTIR D'AUTRES DISPOSITIFS

0) Utiliser les programmes de routine KERNAL OPEN et CHKIN.

1) Appeler ce programme (avec une instruction JSR).

2) Stocker les données.

EXEMPLE:

```
JSR CHRIN
STA DATA
```

B-5. Nom de fonction: CHROUT

But: Sortie d'un caractère
Adresse d'appel: $FFD2 (hexadécimal), 65490 (décimal)
Registres de communication: .A
Programmes de routine de préparation: (CHKOUT, OPEN)
Retours d'erreur: 0 (Voir READST)
Conditions de pile: 8+
Registres affectés: .A

Description: Ce programme de routine fait sortir un caractère vers un canal déjà ouvert. Utiliser les programmes de routine KERNAL OPEN et CHKOUT pour établir le canal de sortie avant d'appeler ce programme. Si l'on omet cet appel, les données sont envoyées au dispositif de sortie implicite (numéro 3 qui est l'écran). L'octet de données à sortir est chargé dans l'accumulateur et le présent programme est appelé. Les données sont ensuite envoyées au dispositif de sortie spécifié. Le canal reste ouvert après l'appel.

REMARQUE: Quand on utilise ce programme de routine, veiller à envoyer les données à un dispositif série spécifique, sinon elles sont envoyées à tous les canaux de sortie ouverts du bus. Si cette condition est indésirable, il faut fermer tous les canaux de sortie ouverts du bus série, en dehors du canal de destination prévu, par un appel au programme KERNAL CLRCHN.

Utilisation:

0) Utiliser le programme de routine KERNAL CHKOUT, si nécessaire (voir description ci-dessous).
1) Charger les données à sortir dans l'accumulateur.
2) Appeler ce programme.

EXEMPLE:

```
;REPRODUIT L'INSTRUCTION BASIC CMD 4,"A";
LDX #4                              ;FICHIER LOGIQUE N° 4
JSR CHKOUT                          ;COUPE LE CANAL OUVERT
LDA #'A
JSR CHROUT                          ;ENVOIE LE CARACTÈRE
```

B-6. Nom de fonction: CIOUT

But: Transmission d'un octet par le bus série
Adresse d'appel: $FFA8 (hexadécimal), 65448 (décimal)
Registres de communication: .A
Programmes de routine de préparation: LISTEN, */SECOND/*
Retours d'erreur: (Voir READST)
Conditions de pile: 5
Registres affectés: Aucun

Description: Ce programme de routine sert à envoyer des informations à d'autres dispositifs sur le bus série. Un appel à ce programme met un octet de données sur le bus série, par l'établissement d'une liaison série intégrale. Avant d'appeler ce programme, on doit utiliser le programme KERNAL LISTEN pour ordonner à un dispositif sur le bus série de se préparer à recevoir des données. (Si un dispositif doit avoir une adresse secondaire, il faut aussi l'envoyer avec le programme KERNAL SECOND.) L'accumulateur est chargé d'un octet pour établir la liaison de données sur le bus série. Un dispositif doit être à l'écoute, sinon le mot d'état donne un dépassement de temps. Ce programme tamponne toujours un caractère. Le programme garde le caractère précédent à renvoyer. Quand on appelle le programme KERNAL UNLSN pour terminer la transmission des données, le caractère tamponné est envoyé avec un jeu "fin ou identification" (EOI). La commande UNLSN est ensuite envoyée au dispositif.

Utilisation:

0) Utiliser le programme KERNAL LISTEN (et le programme SECOND, le cas échéant).
1) Charger l'accumulateur avec un octet de données.
2) Appeler ce programme pour envoyer l'octet de données.

EXEMPLE:

```
LDA#'X                                ;ENVOIE UN X AU BUS SÉRIE
JSR CIOUT
```

B-7. Nom de fonction: CINT

Initialise l'éditeur d'écran et la microplaquette vidéo 6567

Adresse d'appel: $FF81 (hexadécimal), 65409 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 4
Registres affectés: .A, .X, .Y

Description: Ce programme de routine établit la microplaquette du contrôleur vidéo 6567 du Commodore 64 pour le fonctionnement normal. L'éditeur d'écran de KERNAL est également initialisé. Ce programme doit être appelé par une cartouche de programme Commodore 64.

Utilisation:

1) Appeler ce programme.

EXEMPLE:

```
JSR CINT
JMP RUN                               ; COMMENCE L'EXÉCUTION
```

B-8. Nom de fonction: CLALL

But: Fermeture de tous les fichiers
Adresse d'appel: $FFE7 (hexadécimal), 65511 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 11
Registres affectés: .A, .X

Description: Ce programme de routine ferme tous les fichiers ouverts. Quand on appelle ce programme, les pointeurs de la table de fichiers ouverts sont remis à l'état initial et ferment tous les fichiers. Le programme de routine CLRCHN est aussi appelé automatiquement pour remettre les canaux d'entrée/sortie à l'état initial.

Utilisation:

1) Appeler ce programme.

EXEMPLE:

```
JSR CLALL ;FERME TOUS LES FICHIERS ET CHOISIT TOUS LES CANAUX
D'ENTRÉE/ SORTIE IMPLICITE
JMP RUN ;COMMENCE L'EXÉCUTION
```

B-9. Nom de fonction: CLOSE

But: Fermeture d'un fichier logique
Adresse d'appel: $FFC3 (hexadécimal), 65475 (décimal)
Registres de communication: .A
Programmes de routine de préparation: Aucun
Retours d'erreur: 0,240 (Voir READST)
Conditions de pile: 2+
Registres affectés: .A, .X, .Y

Description: Ce programme de routine sert à fermer un fichier logique après l'exécution de toutes les opérations d'entrée/sortie dans ce fichier. On appelle ce programme après avoir chargé l'accumulateur du numéro du fichier logique à fermer (nombre identique à celui utilisé quand on a ouvert le fichier avec le programme OPEN).

Utilisation:

1) Charger l'accumulateur avec le numéro du fichier logique à fermer.
2) Appeler ce programme.

EXEMPLE:

```
;CLOSE 15
LDA #15
JSR CLOSE
```

B-10. Fonction de nom: CLRCHN

But: Effacement des canaux d'entrée/sortie
Adresse d'appel: $FFCC (hexadécimal), 65484 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: Aucun
Retours d'erreur:
Conditions de pile: 9
Registres affectés: .A, .X

Description: On appelle ce programme de routine pour effacer tous les canaux ouverts et rétablir les canaux d'entrée/sortie à leurs valeurs implicites initiales. On l'appelle généralement après l'ouverture d'autres canaux d'entrée/sortie (magnétocassette ou unité de disque) qui servent à des opérations d'entrée/ sortie. Le dispositif d'entrée implicite est 0 (clavier). Le dispositif de sortie implicite est le numéro 3 (écran du Commodore 64).

Si l'un des canaux à fermer est vers l'accès série, un signal UNTALK est d'abord envoyé pour effacer le canal d'entrée. Un signal UNLISTEN peut aussi être envoyé pour effacer le canal de sortie. Si l'on n'appelle pas ce programme de routine (et si on laisse d'autres dispositifs d'écoute en fonction sur le bus série), plusieurs dispositifs peuvent recevoir simultanément les mêmes données du Commodore 64. On peut profiter de cette situation pour ordonner à l'imprimante d'émettre (TALK) et à l'unité de disque d'écouter (LISTEN). On peut ainsi imprimer directement un fichier sur disque.

Ce programme est automatiquement appelé pendant l'exécution du programme KERNAL CLALL.

Utilisation:

1) Appeler ce programme à l'aide d'une instruction JSR.

EXEMPLE:

```
JSR CLRCHN
```

B-11. Nom de fonction: GETIN

But: Extraction d'un caractère
Adresse d'appel: $FFE4 (hexadécimal), 65508 (décimal)
Registres de communication: .A
Programmes de routine de préparation: CHKIN, OPEN
Retours d'erreur: (Voir READST)
Conditions de pile: 7+
Registres affectés: .A (.X, .Y)

Description: Si le canal correspond au clavier, ce sous-programme extrait un caractère de la file d'attente de clavier et le retourne sous forme de valeur ASCII dans l'accumulateur. Si la file d'attente est vide, la valeur retournée dans l'accumulateur est zéro. Les caractères sont automatiquement placés dans la file d'attente par un programme d'exploration de clavier à interruption qui appelle le programme de routine SCNKEY. Le tampon de clavier peut contenir jusqu'à dix caractères. Quand le tampon est plein, les autres caractères sont ignorés jusqu'à ce qu'un caractère au moins soit retiré de la file. Si le canal correspond au RS-232, le registre .A est alors seul utilisé et un seul caractère est retourné. Voir READST pour vérifier la validité. Si le canal correspond à l'accès série, au magnétocassette ou à l'écran, appeler le programme de routine BASIN.

Utilisation:

1) Appeler ce programme à l'aide d'une instruction JSR.
2) Vérifier s'il y a un zéro dans l'accumulateur (tampon vide).
3) Traiter les données.

EXEMPLE:

```
;ATTENDRE UN CARACTÈRE
WAIT JSR GETIN
CMP #0
BEQ WAIT
```

B-12. Nom de fonction: IOBASE

But: Définition de la page de mémoire d'entrée/sortie
Adresse d'appel: $FFF3 (hexadécimal), 65523 (décimal)
Registres de communication: .X, .Y
Programmes de routine de préparation: Aucun
Retours d'erreur:
Conditions de pile: 2
Registres affectés: .X, .Y

Description: Ce programme de routine fixe les registres X et Y à l'adresse de la section de mémoire où se trouve le dispositif d'entrée/sortie en topographie de mémoire. On peut ensuite utiliser cette adresse avec un décalage pour accéder aux dispositifs d'entrée/sortie en topographie de mémoire, dans le Commodore 64. Le décalage correspond au nombre de positions, depuis le début de la page où se trouve le registre d'entrée/sortie désiré. Le registre .X contient l'octet d'adresse de poids faible et le registre .Y l'octet d'adresse de poids fort.

Ce programme de routine permet la compatibilité entre le Commodore 64, le VIC-20 et les futurs modèles de Commodore 64. Si les positions d'entrée/sortie d'un programme en langage machine sont fixées par un appel à ce programme, elles doivent rester compatibles avec les versions ultérieures du Commodore 64, du KERNAL et du BASIC.

Utilisation:

1) Appeler ce programme à l'aide de l'instruction JSR.
2) Stocker les registres .X et .Y dans des positions consécutives.
3) Charger le registre .Y avec le décalage.
4) Accéder à cette position d'entrée/sortie.

EXEMPLE:

```
;FIXE LE REGISTRE DE DIRECTION DE DONNÉES DE L'ACCÈS D'UTILISATEUR
À 0 (ENTRÉE)
JSR IOBASE
STX POINT ;FIXE LES REGISTRES DE BASE
STY POINT+1
LDY #2
LDA #0 ;DÉCALAGE POUR DDR DE L'ACCÈS D'UTILISATEUR
STA (POINT), Y;FIXE DDR À 0
```

B-13. Nom de fonction: IOINIT

But: Initialisation des dispositifs d'entrée/sortie
Adresse d'appel: $FF84 (hexadécimal), 65412 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: Aucun
Retours d'erreur:
Conditions de pile: Aucun
Registres affectés: .A, .X, .Y

Description: Ce programme de routine initialise les programmes et dispositifs d'entrée/sortie. On l'appelle normalement dans le cadre de la procédure d'initialisation d'une cartouche de programme Commodore 64.

EXEMPLE:

```
JSR IOINIT
```

B-14. Nom de fonction: LISTEN

But: Ordre d'écouter donné à un dispositif sur le bus série
Adresse d'appel: $FFB1 (hexadécimal), 65457 (décimal)
Registres de communication: .A
Programmes de routine de préparation: Aucun
Retours d'erreur: (Voir READST)
Conditions de pile: Aucune
Registres affectés: .A

Description: Ce programme de routine ordonne à un dispositif sur le bus série de recevoir des données. Charger l'accumulateur d'un numéro de dispositif entre 0 et 31 avant d'appeler le programme. Le programme LISTEN effectue des opérations logiques OU, bit par bit, sur le numéro pour le convertir en adresse d'écoute, puis transmettre ces données comme commande, sur le bus série. Le dispositif spécifié passe ensuite en mode d'écoute pour recevoir les informations.

Utilisation

1) Charger l'accumulateur du numéro du dispositif qui doit écouter.
2) Appeler ce programme à l'aide de l'instruction JSR.

EXEMPLE:

```
;COMMANDE AU DISPOSITIF #8 D'ÉCOUTER
LDA #8
JSR LISTEN
```

B-15. Nom de fonction: LOAD

But: Chargement de la mémoire vive à partir d'un dispositif
Adresse d'appel: $FFD5 (hexadécimal), 65493 (décimal)
Registres de communication: .A, .X, .Y
Programmes de routine de préparation: SETLFS, SETNAM
Retours d'erreur: 0,4,5,8,9, READST
Conditions de pile: Aucune
Registres affectés: .A, .X, .Y

Description: Ce programme de routine charge directement les octets de données de tout dispositif d'entrée dans la mémoire du Commodore 64. On peut aussi l'utiliser dans une opération de vérification, pour comparer les données d'un dispositif à celles déjà en mémoire, tout en laissant inchangées les données stockées dans la mémoire vive.

Mettre l'accumulateur (.A) à 0 pour une opération de chargement ou à 1 pour une vérification. Si le dispositif d'entrée est ouvert par une adresse secondaire (SA) de 0, les informations de titre du dispositif sont ignorées. Dans ce cas, les registres .X et .Y doivent contenir l'adresse de départ pour le chargement. Si l'on désigne le dispositif avec une adresse secondaire de 1, les données sont alors chargées dans la mémoire à partir de la position spécifiée par l'en-tête. Ce programme retourne l'adresse de la position de mémoire vive la plus haute qui est chargée.

Avant de pouvoir utiliser ce programme, on doit appeler les programmes de routine KERNAL SETLFS et SETNAM.

REMARQUE: On ne peut pas charger à partir du clavier (0), du RS-232 (2) ou de l'écran (3).

Utilisation

0) Appeler les programmes de routine SETLFS et SETNAM. Si l'on désire un chargement déplacé, envoyer une adresse secondaire de 0 à l'aide du programme de routine SETLFS.
1) Fixer le registre .A à 0 pour le chargement et à 1 pour la vérification.
2) Si l'on désire un chargement déplacé, fixer les registres .X et .Y à l'adresse de départ du chargement.
3) Appeler le programme à l'aide de l'instruction JSR.

EXEMPLE:

```
        ;CHARGEMENT D'UN FICHIER DE CASSETTE
        LDA   #DISPOSITIF1   ;FIXE LE NUMÉRO DE DISPOSITIF
        LDX   #FICHIER N°    ;FIXE LE NUMÉRO DE FICHIER LOGIQUE
        LDY   CMD1           ;FIXE L'ADRESSE SECONDAIRE
        JSR   SETLFS
        LDA   #NOM1-NOM      ;CHARGE .A AVEC LE NOMBRE DE
                             ;CARACTÈRES DANS LE NOM DE FICHIER
        LDX   #<NOM          ;CHARGE .X ET .Y AVEC
                             ;L'ADRESSE DU
        LDY   #>NOM          ;NOM DE FICHIER
        JSR   SETNAM
        LDA   #0             ;FIXE L'INDICATEUR POUR UN
                             ;CHARGEMENT
        LDX   #$FF           ;AUTRE DÉPART
        LDY   #$FF
        JSR   LOAD
        STX   VARTAB         ;FIN DE CHARGEMENT
        STY   VARTAB+1
        JMP   START
NOM     .BYT  'NOM DE FICHIER'
NOM 1   ;
```

B-16. Nom de fonction: MEMBOT

But: Établissement du bas de la mémoire

Adresse d'appel: $FF9C (hexadécimal), 65436 (décimal)

Registres de communication: .X, .Y

Programmes de routine de préparation: Aucun

Retours d'erreur: Aucun

Conditions de pile: Aucune

Registres affectés: .X, .Y

Description: Ce programme de routine sert à établir le bas de la mémoire. Si le bit de report d'accumulateur est fixé quand on appelle ce programme, un pointeur de l'octet de poids le plus faible de la mémoire vive est retourné dans les registres .X et .Y. Avec le Commodore 64 non étendu, la valeur initiale de ce pointeur est $0800 (2048 en décimal). Si le bit de report d'accumulateur est égal à 0 à l'appel de ce programme, les valeurs des registres .X et .Y sont transférées respectivement aux octets de poids faible et de poids fort du pointeur, au début de la mémoire vive.

Utilisation:

LECTURE DU BAS DE LA MÉMOIRE VIVE

1) Fixer le report.
2) Appeler ce programme.

ÉTABLISSEMENT DU BAS DE LA MÉMOIRE

1) Effacer le report.
2) Appeler ce programme.

EXEMPLE:

```
;DÉPLACEMENT DU BAS DE LA MÉMOIRE À LA PAGE 1
SEC          ;LECTURE DU BAS DE LA MÉMOIRE
JSR MEMBOT
INY
CLC          ;ÉTABLISSEMENT DU BAS DE LA MÉMOIRE À LA NOUVELLE
             VALEUR
JSR MEMBOT
```

B-17. Nom de fonction: MEMTOP

But: Établissement du haut de la mémoire vive
Adresse d'appel: $FF99 (hexadécimal), 65433 (décimal)
Registres de communication: .X, .Y
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: .X, .Y

Description: Ce programme de routine sert à fixer le haut de la mémoire vive. Quand on appelle ce programme, avec le bit de report de l'accumulateur fixé, le pointeur de la mémoire vive est chargé dans les registres .X et .Y. Quand on appelle ce programme avec le bit de report d'accumulateur à zéro, les contenus des registres .X et .Y sont chargés dans le haut du pointeur de mémoire pour changer celui-ci.

EXEMPLE:

```
;DÉSAFFECTATION DU TAMPON RS-232
SEC
JSR MEMTOP ;LECTURE DU HAUT DE LA MÉMOIRE
DEX
CLC
JSR MEMTOP ;ÉTABLISSEMENT DU NOUVEAU HAUT DE MÉMOIRE
```

B-18. Nom de fonction: OPEN

But: Ouverture d'un fichier logique
Adresse d'appel: $FFC0 (hexadécimal), 65472 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: SETLFS, SETNAM
Retours d'erreur: 1,2,4,5,6,240, READST
Conditions de pile: Aucun
Registres affectés: .A, .X, .Y

Description: Ce programme de routine sert à ouvrir un fichier logique. Quand le fichier logique est établi, il peut servir à des opérations d'entrée/sortie. On utilise ce programme avec la plupart des programmes de routine d'entrée/sortie de KERNAL pour créer des fichiers logiques de travail. Il n'y a aucun argument à établir pour utiliser ce programme, mais il faut appeler au préalable les programmes de routine KERNAL SETLFS et SETNAM.

Utilisation:

0) Appeler le programme SETLFS.
1) Appeler le programme SETNAM.
2) Appeler ce programme.

EXEMPLE:

Ce programme est une mise en oeuvre de l'instruction BASIC: OPEN 15,8,15,"I/O"

```
        LDA # NOM2-NOM    ;LONGUEUR DU NOM DE FICHIER
                          ;POUR SETLFS
        LDY # >NOM        ;ADRESSE DU NOM DE FICHIER
        LDX # <NOM
        JSR SETNAM
        LDA #15
        LDX #8
        LDY #15
        JSR SETLFS
        JSR OPEN
NOM     .BYT 'I/O'
NOM2
```

B-19. Nom de fonction: PLOT

But: Établissement de la position du curseur
Adresse d'appel: $FFF0 (hexadécimal), 65520 (décimal)
Registres de communication: .A, .X, .Y
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: .A, .X, .Y

Description: Quand on appelle ce programme de routine, avec l'indicateur de report d'accumulateur fixé, on charge la position présente du curseur sur l'écran (en coordonnées X et Y), dans les registres .Y et .X. Y correspond au numéro de colonne de la position du curseur (6 à 39) et X au numéro de rangée de cette position (0 à 24). Un appel avec un bit de report à zéro déplace le curseur à X, Y, tel que le déterminent les registres .Y et .X.

Utilisation:

LECTURE DE LA POSITION DU CURSEUR

1) Mettre l'indicateur de report à un.
2) Appeler ce programme.
3) Extraire les positions X et Y respectivement des registres .Y et .X.

ÉTABLISSEMENT DE LA POSITION DU CURSEUR

1) Mettre l'indicateur de report à zéro.
2) Fixer les registres .Y et .X à la position désirée du curseur.
3) Appeler ce programme.

EXEMPLE:

```
; DÉPLACEMENT DU CURSEUR À LA RANGÉE 10, COLONNE 5 (5,10)
LDX #10
LDY #5
CLC
JSR PLOT
```

B-20. Nom de fonction: RAMTAS

But: Exécution de la vérification de mémoire vive

Adresse d'appel: $FF87 (hexadécimal), 65415 (décimal)

Registres de communication: .A, .X, .Y

Programmes de routine de préparation: Aucun

Retours d'erreur: Aucun

Conditions de pile: 2

Registres affectés: .A, .X, .Y

Description: Ce programme de routine sert à vérifier la mémoire RAM et à fixer les pointeurs de haut et de bas de mémoire en conséquence. Il efface aussi les positions $0000 à $0101 et $0200 à $03FF. Il attribue le tampon de cassette et fixe la base d'écran à $0400. Dans des conditions normales, on appelle ce programme dans le cadre du processus d'initialisation d'une cartouche de programme Commodore 64.

EXEMPLE:

```
JSR RAMTAS
```

B-21. Nom de fonction: RDTIM

But: Lecture d'horloge du système

Adresse d'appel: $FFDE (hexadécimal), 65502 (décimal)

Registres de communication: .A, .X, .Y

Programmes de routine de préparation: Aucun

Retours d'erreur: Aucun

Conditions de pile: 2

Registres affectés: .A, .X, .Y

Description: Ce programme de routine sert à lire l'horloge du système. La précision de l'horloge est de 1/60 de seconde. Le programme retourne trois octets. L'accumulateur contient l'octet le plus significatif, le registre d'index X contient l'octet le plus significatif suivant et le registre d'index X, l'octet le moins significatif.

EXEMPLE:

```
JSR RDTIM
STY HEURE
STX HEURE+1
STA HEURE+2
 . . .
HEURE *=*+3
```

B-22. Nom de fonction: READST

But: Lecture du mot d'état
Adresse d'appel: $FFB7 (hexadécimal), 65463 (décimal)
Registres de communication: .A
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: .A

Description: Ce programme de routine retourne l'état présent des dispositifs d'entrée/sortie dans l'accumulateur. Le programme est généralement appelé après une nouvelle communication avec un dispositif d'entrée/sortie. Le programme donne des indications sur l'état du dispositif ou les erreurs qui se sont produites pendant l'opération d'entrée/sortie.

Les bits retournés dans l'accumulateur contiennent les informations suivantes: (voir table ci-dessous)

POSITION DE BIT D'ÉTAT	VALEUR NUMÉRIQUE D'ÉTAT	LECTURE CASSETTE	LECTURE/ ÉCRITURE SÉRIE	VÉRIFICATION +CHARGEMENT CASSETTE
0	1		Délai d'attente d'écriture	
1	2		Délai d'attente de lecture	
2	4	Bloc court		Bloc court
3	8	Bloc long		Bloc long
4	16	Erreur de lecture irrémédiable		Toute erreur d'assortiment
5	32	Erreur de total de contrôle		Erreur de total de contrôle
6	64	Fin de fichier	Ligne de fin ou identification	
7	–128	Fin de bande	Dispositif absent	Fin de bande

Utilisation:

1) Appeler ce programme.
2) Décoder les informations dans le registre .A, telles qu'elles se rapportent au programme.

EXEMPLE:

```
;VÉRIFICATION DE LA FIN DE FICHIER PENDANT LA LECTURE
JSR READST
AND #64          ;VÉRIFICATION DE BIT DE FIN DE FICHIER (EOF)
BNE EOF          ;BRANCHEMENT SUR FIN DE FICHIER
```

B-23. Nom de fonction: RESTOR

But: Rétablissement du système implicite et des vecteurs d'interruption
Adresse d'appel: $FF8A (hexadécimal), 65418 (décimal)
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: .A, .X, .Y

Description: Ce programme de routine rétablit les valeurs implicites de tous les vecteurs de système utilisés dans les programmes de routine et interruptions BASIC et KERNAL. (Voir la topographie de mémoire pour les contenus de vecteurs implicites.) Le programme de routine KERNAL VECTOR sert à lire et à modifier les vecteurs individuels de système.

Utilisation:

1) Appeler ce programme.

EXEMPLE:

```
JSR RESTOR
```

B-24. Nom de fonction: SAVE

But: Sauvegarde de la mémoire dans un dispositif
Adresse d'appel: $FFD8 (hexadécimal), 65496 (décimal)
Registres de communication: .A, .X, .Y
Programmes de routine de préparation: SETLFS, SETNAM
Retours d'erreur: 5,8,9 (Voir READST)
Conditions de pile: Aucune
Registres affectés: .A, .X, .Y

Description: Ce programme de routine sauvegarde une section de mémoire. La mémoire est sauvegardée d'une adresse indirecte, à la page 0 spécifiée par l'accumulateur, à l'adresse stockée dans les registres .X et .Y. Elle est ensuite envoyée à un fichier logique dans un dispositif d'entrée/sortie. Utiliser les programmes de routine SETLFS et SETNAM avant d'appeler ce programme. Il n'est cependant pas nécessaire d'avoir un nom de fichier pour la sauvegarde sur le dispositif 1 (magnétocassette Datasette™). Toute tentative de sauvegarde sur un autre dispositif sans utilisation d'un nom de fichier se traduit par une erreur.

REMARQUE: On ne peut pas sauvegarder vers le dispositif 0 (clavier), le dispositif 2 (RS-232) et le dispositif 3 (écran). En cas de tentative, il se produit une erreur et la sauvegarde est interrompue.

Utilisation:

0) Appeler les programmes de routine SETLFS et SETNAM, sauf si l'on veut exécuter une sauvegarde sans nom de fichier vers le magnétocassette.
1) Charger deux positions consécutives à la page 0 avec un pointeur au début de la sauvegarde (en format 6502 standard, octet de poids faible en premier et octet de poids fort ensuite).
2) Charger l'accumulateur, avec le décalage à octet simple de page zéro, au pointeur.
3) Charger les registres .X et .Y respectivement avec l'octet de poids faible et l'octet de poids fort de la position de fin de sauvegarde.
4) Appeler ce programme.

EXEMPLE:

```
LDA                   ;DISPOSITIF=1:CASSETTE
JSR SETLFS
LDA #0                ;PAS DE NOM DE FICHIER
JSR SETNAM
LDA PROG              ;CHARGER L'ADRESSE DE DÉPART DE LA
                      SAUVEGARDE
STA TXTTAB            ;(OCTET DE POIDS FAIBLE)
LDA PROG+1
STA TXTTAB+1          ;(OCTET DE POIDS FORT)
LDX VARTAB+1          ;CHARGER .X AVEC L'OCTET DE POIDS FAIBLE DE LA
                      FIN DE ;SAUVEGARDE
LDY VARTAB+1          ;CHARGER .Y AVEC L'OCTET DE POIDS FORT
LDA #<TXTTAB          ;CHARGER L'ACCUMULATEUR AVEC LE DÉCALAGE
                      DE PAGE 0
JSR SAVE
```

B-25. Nom de fonction: SCNKEY

But: Exploration du clavier
Adresse d'appel: $FF9F (hexadécimal), 65439 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: IOINIT
Retours d'erreur: Aucun
Conditions de pile: 5
Registres affectés: .A, .X, .Y

Description: Ce programme de routine explore le clavier du Commodore 64 et vérifie les touches manipulées. Ce programme est identique à celui appelé par le programme d'interruption. Si l'on a appuyé sur une touche, sa valeur ASCII est mise dans la file d'attente du clavier. Ce programme n'est appelé que si l'interruption IRQ normale est ignorée.

Utilisation:

1) Appeler ce programme.

EXEMPLE

```
GET   JSR SCNKEY      ;EXPLORE LE CLAVIER
      JSR GETIN       ;EXTRAIT UN CARACTÈRE
      CMP #0          ;EST-IL NUL?
      BEQ GET         ;OUI . . . EXPLORE DE NOUVEAU
      JSR CHROUT      ;IMPRIME LE CARACTÈRE
```

B-26. Nom de fonction: SCREEN

But: Retour du format d'écran
Adresse d'appel: $FFED (hexadécimal), 65517 (décimal)
Registres de communication: .X, .Y
Programmes de routine de préparation: Aucun
Conditions de pile: 2
Registres affectés: .X, .Y

Description: Ce programme de routine retourne le format de l'écran, c'est-à-dire 40 colones dans le registre .X et 25 lignes dans le registre .Y. On peut l'utiliser pour déterminer dans quelle machine se déroule l'exécution d'un programme. Cette fonction a été mise en oeuvre dans le Commodore 64 pour maintenir la compatibilité ascendante des programmes de l'utilsateur.

Utilisation:

1) Appeler ce programme.

EXEMPLE:

```
JSR SCREEN
STX MAXCOL
STY MAXROW
```

B-27. Nom de fonction: SECOND

But: Envoi d'une adresse secondaire pour LISTEN
Adresse d'appel: $FF93 (hexadécimal), 65427 (décimal)
Registres de communication: .A
Programmes de routine de préparation: LISTEN
Retours d'erreur: Voir READST
Conditions de pile: 8
Registres affectés: .A

Description: Ce programme de routine sert à envoyer une adresse secondaire à un dispositif d'entrée/sortie après un appel au programme LISTEN. Le dispositif est commandé par LISTEN. On ne peut pas utiliser ce programme pour envoyer une adresse secondaire, après un appel au programme de routine TALK.

On utilise généralement une adresse secondaire pour donner des informations d'établissement à un dispositif, avant les opérations d'entrée/sortie.

Quand on doit envoyer une adresse secondaire à un dispositif sur le bus série, on doit d'abord procéder à une opération logique OU de l'adresse avec $60.

Utilisation:

1) Charger l'accumulateur avec l'adresse secondaire à envoyer.
2) Appeler ce programme.

EXEMPLE:

```
;ADRESSAGE DU DISPOSITIF # 8 AVEC LA COMMANDE (ADRESSE
SECONDAIRE) # 15
LDA #8
JSR LISTEN
LDA #15
JSR SECOND
```

B-28. Nom de fonction: SETLFS

But: Établissement d'un fichier logique
Adresse d'appel: $FFBA (hexadécimal), 65466 (décimal)
Registres de communication: .A, .X, .Y
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: Aucun

Description: Ce programme de routine fixe le numéro de fichier logique, l'adresse de dispositif et l'adresse secondaire (numéro de commande) pour les autres programmes de routine KERNAL.

Le système utilise le numéro de fichier logique comme indicatif de la table de fichier créée par le programme de routine d'ouverture (OPEN) de fichier. Les adresses de dispositif peuvent aller de 0 à 31. Le Commodore 64 utilise les codes suivants pour les dispositifs CBM indiqués ci-dessous:

ADRESSE	DISPOSITIF
0	Clavier
1	Magnétocassette Datasette™ N° 1
2	Dispositif RS-232C
3	Affichage du TRC
4	Imprimante de bus série
8	Unité de disque de bus série CBM

À partir du numéro 4, les dispositifs se trouvent automatiquement sur le bus série.

Une commande au dispositif est envoyée comme adresse secondaire sur le bus série après l'envoi d'un numéro de dispositif pendant la séquence de mise en liaison série. Si l'on ne doit envoyer aucune adresse secondaire, le registre d'index .Y doit être fixé à 255.

Utilisation:

1) Charger l'accumulateur avec le numéro du fichier logique.
2) Charger le registre d'index .X avec le numéro de dispositif.
3) Charger le registre d'index .Y avec la commande.

EXEMPLE:

POUR LE FICHIER LOGIQUE 32 ET LE DISPOSITIF # 4, AUCUNE COMMANDE:

```
LDA #32
LDX #4
LDY #255
JSR SETLFS
```

B-29. Nom de fonction: SETMSG

But: Commande de la sortie de message du système
Adresse d'appel: $FF90 (hexadécimal), 65424 (décimal)
Registres de communication: .A
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: .A

Description: Ce programme de routine commande l'impression des messages d'erreur et de commande par le KERNAL. On peut choisir l'impression des messages d'erreur ou des messages de commande en fixant l'accumulateur à l'appel du programme. "FILE NOT FOUND" (Fichier non trouvé) est un exemple de message d'erreur. "PRESS PLAY ON CASSETTE" (Appuyer sur la touche de lecture du magnétocassette) est un exemple de message de commande.

Les bits 6 et 7 de cette valeur déterminent la provenance du message. Si le bit 7 est à 1, l'un des messages d'erreur du KERNAL est imprimé. Si le bit 6 est à 1, les messages de commande sont imprimés.

Utilisation:

1) Fixer l'accumulateur à la valeur désirée.
2) Appeler ce programme.

EXEMPLE:

```
LDA #40
JSR SETMSG        ;MET LES MESSAGES DE COMMANDE EN FONCTION
LDA #$80
JSR SETMSG        ;MET LES MESSAGES D'ERREUR EN FONCTION
LDA #0
JSR SETMSG        ;COUPE TOUS LES MESSAGES DE KERNAL
```

B-30. Nom de fonction: SETNAM

But: Établissement d'un nom de fichier
Adresse d'appel: $FFBD (hexadécimal), 65469 (décimal)
Registres de communication: .A, .X, .Y
Programmes de routine de préparation: Aucun
Conditions de pile: Aucune
Registres affectés: Aucun

Description: Ce programme de routine sert à établir le nom de fichier pour les programmes de routine OPEN, SAVE et LOAD. On doit charger la longueur du nom de fichier dans l'accumulateur. Charger l'adresse du nom de fichier dans les registres .X et .Y, en format 6502 standard avec octet de poids faible/octet de poids fort. L'adresse peut correspondre à toute adresse de mémoire valide dans le système où est stocké une chaîne de caractères pour le nom de fichier. Si l'on ne désire aucun nom de fichier, fixer l'accumulateur à 0, c'est-à-dire à une longueur de fichier nulle. Dans ce cas, on peut fixer les registres .X et .Y à toute adresse de mémoire.

Utilisation:

1) Charger l'accumulateur avec la longueur du nom de fichier.
2) Charger le registre d'index .X avec l'adresse basse du nom de fichier.
3) Charger le registre d'index .Y avec l'adresse haute du nom de fichier.
4) Appeler ce programme.

EXEMPLE:

```
LDA #NOM2-NOM     ;CHARGE LA LONGUEUR DU NOM DE FICHIER
LDX #<NOM         ;CHARGE L'ADRESSE DU NOM DE FICHIER
LDY #>NOM
JSR SETNAM
```

B-31. Nom de fonction: SETTIM

But: Réglage de l'horloge du système
Adresse d'appel: $FFDB (hexadécimal), 65499 (décimal)
Registres de communication: .A, .X, .Y
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: Aucun

Description: Une horloge de système est maintenue par un programme de routine d'interruption qui la met à jour chaque 1/60 de seconde. L'horloge, avec trois octets de long, peut compter jusqu'à concurrence de 5,184,000 soixantièmes de seconde, soit 24 heures. À ce point, l'horloge se remet à zéro. Avant d'appeler ce programme pour régler l'horloge, l'accumulateur doit contenir l'octet le plus significatif, le registre d'index .X, l'octet le plus significatif suivant et le registre d'index .Y, l'octet le moins significatif du réglage initial de l'heure (en 1/60 de seconde).

Utilisation:

1) Charger l'accumulateur avec l'octet le plus significatif du nombre à 3 octets pour régler l'horloge.
2) Charger le registre .X avec l'octet suivant.
3) Charger le registre .Y avec l'octet le moins significatif.
4) Appeler ce programme.

EXEMPLE:

```
;RÉGLAGE DE L'HORLOGE À 10 MINUTES = 3600 1/60 DE SECONDE
LDA #0              ;OCTET LE PLUS SIGNIFICATIF
LDX #>3600
LDY #<3600          ;OCTET LE MOINS SIGNIFICATIF
JSR SETTIM
```

B-32. Nom de fonction: SETTMO

But: Réglage de l'indicateur de délai d'attente de la carte de bus IEEE
Adresse d'appel: $FFA2 (hexadécimal), 65442 (décimal)
Registres de communication: .A
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: Aucun

> **REMARQUE:** Ce programme ne s'utilise qu'avec une carte complémentaire IEEE.

Description: Ce programme de routine fixe l'indicateur de délai d'attente pour le bus IEEE. Quand l'indicateur de délai d'attente est réglé, le Commodore 64 attend un dispositif sur l'accès IEEE pendant 64 millisecondes. Si le dispositif ne répond pas au signal valide d'adresse de données (DAV) du Commodore 64 pendant cette durée, l'ordinateur identifie un état d'erreur et sort de la séquence d'établissement de liaison. À l'appel de ce programme, quand l'accumulateur contient un zéro au bit 7, les délais d'attente sont validés; ils sont invalidés s'il y a un 1 dans le bit 7.

> **REMARQUE:** Le Commodore 64 utilise la caractéristique de délai d'attente pour signaler qu'un fichier de disque n'a pas été trouvé dans une tentative d'ouverture (OPEN) d'un fichier (avec carte IEEE seulement).

Utilisation:

ÉTABLISSEMENT DE L'INDICATEUR DE DÉLAI D'ATTENTE

1) Mettre le bit 7 de l'accumulateur à 0.
2) Appeler ce programme.

REMISE DE L'INDICATEUR DE DÉLAI D'ATTENTE À L'ÉTAT INITIAL

1) Mettre le bit 7 de l'accumulateur à 1.
2) Appeler ce programme.

EXEMPLE:

```
;INVALIDATION DU DÉLAI D'ATTENTE
LDA #0
JSR SETTMO
```

B-33. Nom de fonction: STOP

But: Vérification d'une pression sur la touche STOP
Adresse d'appel: $FFE1 (hexadécimal), 65505 (décimal)
Registres de communication: .A
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: Aucune
Registres affectés: .A, .X

Description: Si l'on a appuyé sur la touche STOP du clavier pendant un appel UDTIM, celui-ci retourne l'indicateur Z à l'état zéro. En outre, les canaux sont remis à des valeurs implicites. Tous les autres indicateurs restent inchangés. Si l'on n'a pas appuyé sur la touche STOP, l'accumulateur contient alors un octet représentant la dernière rangée d'exploration du clavier. Ce processus permet à l'utilisateur de vérifier aussi certaines autres touches.

Utilisation:

0) Appeler UDTIM avant ce programme de routine.
1) Appeler ce programme.
2) Vérifier l'indicateur zéro.

EXEMPLE:

```
JSR UDTIM ;RECHERCHE POUR ARRÊT
JSR STOP
BNE *+5 ;TOUCHE NON PRESSÉE
JMP READY ;= . . . ARRÊT
```

B-34. Nom de fonction: TALK

But: Commande d'émission à un dispositif sur le bus série
Adresse d'appel: $FFB4 (hexadécimal), 65460 (décimal)
Registres de communication: .A
Programmes de routine de préparation: Aucun
Retours d'erreur: Voir READST
Conditions de pile: 8
Registres affectés: .A

Description: Pour utiliser ce programme de routine, charger d'abord l'accumulateur d'un numéro de dispositif entre 0 et 31. Quand on appelle ce programme, il procède à une réunion logique OU bit par bit, pour convertir ce numéro de dispositif en adresse d'émission. Les données sont ensuite transmises comme commande sur le bus série.

Utilisation:

1) Charger l'accumulateur avec le numéro de dispositif.
2) Appeler ce programme.

EXEMPLE:

```
;ORDRE D'ÉMISSION AU DISPOSITIF # 4
LDA #4
JSR TALK
```

B-35. Nom de fonction: TKSA

But: Envoi d'une adresse secondaire à un dispositif devant émettre
Adresse d'appel: $FF96 (hexadécimal), 65430 (décimal)
Registres de communication: .A
Programmes de routine de préparation: TALK
Retours d'erreur: Voir READST
Conditions de pile: 8
Registres affectés: .A

Description: Ce programme de routine transmet une adresse secondaire sur le bus série, pour un dispositif devant émettre. Appeler ce programme avec un nombre compris entre 0 et 31 dans l'accumulateur. Le programme envoie ce nombre comme commande d'adresse secondaire par le bus série. On ne peut utiliser ce programme qu'après l'appel du programme TALK. Il est sans effet après un programme LISTEN.

Utilisation:

0) Appeler le programme TALK.
1) Charger l'adresse dans l'accumulateur.
2) Appeler ce programme.

EXEMPLE:

```
;INDIQUE AU DISPOSITIF # 4 D'ÉMETTRE AVEC LA COMMANDE # 7
LDA #4
JSR TALK
LDA #7
JSR TALKSA
```

B-36. Nom de fonction: UDTIM

But: Mise à jour de l'horloge du système
Adresse d'appel: $FFEA (hexadécimal), 65514 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: .A, .X

Description: Ce programme de routine met l'horloge du système à jour. Dans des conditions normales, le programme d'interruption KERNAL normal appelle ce programme chaque 1/60 de seconde. Si le programme de l'utilisateur traite ses propres interruptions, on doit appeler ce programme de routine pour mettre l'heure à jour. En outre, on doit appeler le programme de touche **STOP** si celle-ci doit rester fonctionnelle.

Utilisation:

1) Appeler ce programme.

EXEMPLE:

```
JSR UDTIM
```

B-37. Nom de fonction: UNLSN

But: Envoi d'une commande de désadressage
Adresse d'appel: $FFAE (hexadécimal), 65454 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: Aucun
Retours d'erreur: Voir READST
Conditions de pile: 8
Registres affectés: .A

Description: Ce programme de routine ordonne à tous les dispositifs sur le bus série de cesser de recevoir des données du Commodore 64 (UNLISTEN — désadressage). L'appel de ce programme se traduit par la transmission d'une commande de désadressage sur le bus série. La commande n'affecte que les dispositifs qui avaient précédemment reçu l'ordre d'écoute (LISTEN). On utilise normalement ce programme après que le Commodore 64 ait fini d'envoyer des données aux dispositifs extérieurs. L'envoi d'une commande de désadressage indique aux dispositifs en cours d'écoute de quitter le bus série pour que celui-ci puisse servir à d'autres fins.

Utilisation:

1) Appeler ce programme.

EXEMPLE:

```
JSR UNLSN
```

B-38. Nom de fonction: UNTLK

But: Envoi d'une commande de coupure d'émission
Adresse d'appel: $FFAB (hexadécimal), 65451 (décimal)
Registres de communication: Aucun
Programmes de routine de préparation: Aucun
Retours d'erreur: Voir READST
Conditions de pile: 8
Registres affectés: .A

Description: Ce programme de routine transmet une commande de coupure d'émission sur le bus série. Tous les dispositifs qui avaient précédemment reçu une commande TALK cessent d'envoyer des données à la réception de cette commande.

Utilisation:

1) Appeler ce programme.

EXEMPLE:

```
JSR UNTALK
```

B-39. Nom de fonction: VECTOR

But: Gestion des vecteurs de mémoire vive (RAM)
Adresse d'appel: $FF8D (hexadécimal), 65421 (décimal)
Registres de communication: .X,.Y
Programmes de routine de préparation: Aucun
Retours d'erreur: Aucun
Conditions de pile: 2
Registres affectés: .A, .X, .Y

Description: Ce programme de routine gère toutes les adresses de saut de vecteur du système stockées en mémoire vive. L'appel de ce programme avec la mise à un du bit de report d'accumulateur stocke le contenu courant des vecteurs de mémoire vive dans une liste pointée par les registres .X et .Y. Avec l'appel de ce programme par l'effacement de report, la liste de l'utilisateur pointée par les registres .X et .Y est transférée aux vecteurs de mémoire vive du système. Les vecteurs de mémoire vive sont listés dans la topographie de mémoire.

> **REMARQUE:** L'utilisation de ce programme demande des précautions. Pour l'utiliser, il est préférable de lire d'abord les contenus entiers des vecteurs dans la zone de l'utilisateur, de modifier les vecteurs désirés, puis de recopier les contenus dans les vecteurs du système.

Utilisation:

LECTURE DES VECTEURS DE MÉMOIRE VIVE DU SYSTÈME

1) Fixer le report.
2) Fixer les registres .X et .Y à l'adresse des vecteurs.
3) Appeler ce programme.

CHARGEMENT DES VECTEURS DE MÉMOIRE VIVE DU SYSTÈME

1) Effacer le bit de report.
2) Fixer les registres .X et .Y à l'adresse de la liste de vecteurs en mémoire vive qui doit être chargée.
3) Appeler ce programme.

EXEMPLE:

```
;CHANGEMENT DES PROGRAMMES D'ENTRÉE AU NOUVEAU SYSTÈME
LDX #<UTILISATEUR
LDY #>UTILISATEUR
SEC
JSR VECTEUR          ;LECTURE DES ANCIENS VECTEURS
LDA #<MYINP          ;CHANGEMENT DE L'ENTRÉE
STA UTILISATEUR+10
LDA #>MYINP
STA UTILISATEUR+11
LDX #<UTILISATEUR
LDY #>UTILISATEUR
CLC
JSR VECTEUR          ;MODIFICATION DU SYSTÈME
 . . .
UTILISATEUR *=*+26
```

CODES D'ERREUR

Nous donnons ci-dessous une liste de messages d'erreur pouvant se produire dans l'utilisation des programmes KERNAL. En cas d'erreur pendant un programme KERNAL, le bit de report de l'accumulateur est à un et le numéro du message d'erreur est retourné dans l'accumulateur.

REMARQUE: Certains programmes de routine d'entrée/sortie KERNAL n'utilisent pas ces codes pour les messages d'erreur. À la place, les erreurs sont identifiées avec le programme de routine KERNAL READST.

NUMÉRO	SIGNIFICATION
0	Programme de routine terminé par la touche STOP
1	Trop de fichiers ouverts
2	Fichier déjà ouvert
3	Fichier non ouvert
4	Fichier non trouvé
5	Dispositif absent
6	Le fichier n'est pas un fichier d'entrée
7	Le fichier n'est pas un fichier de sortie
8	Le nom du fichier est manquant
9	Numéro de dispositif interdit
240	Attribution/désaffectation de tampon RS-232 de changement de haut de mémoire

UTILISATION DU LANGAGE MACHINE À PARTIR DU BASIC

Il existe plusieurs méthodes d'utilisation du BASIC et du langage machine avec le Commodore 64; il faut y inclure des instructions spéciales dans le BASIC CBM et des positions clés en machine. Avec le Commodore 64, on peut utiliser les programmes en langage machine à partir du BASIC de cinq manières essentielles qui sont:

1) Instruction SYS de BASIC
2) Fonction USR de BASIC
3) Changement d'un des vecteurs d'entrée/sortie de mémoire vive
4) Changement d'un des vecteurs d'interruption de mémoire vive
5) Changement du programme de routine CHRGET

1) L'instruction **SYS X** de BASIC amène un saut à un sous-programme de langage machine placé à l'adresse X. Le programme doit se terminer par une instruction **RTS** (Retour du sous-programme) pour ramener la commande au BASIC.

   Les paramètres sont généralement communiqués entre le programme de langage machine et le programme BASIC par les instructions PEEK et POKE de BASIC et leurs équivalents en langage machine.

   La commande **SYS** constitue le moyen le plus pratique de combiner le BASIC au langage machine. Les instructions PEEK et POKE facilitent le passage de paramètres multiples. Un programme peut contenir plusieurs instructions SYS, chacune se rapportant à un programme de routine en langage machine différent (ou identique).

2) La fonction **USR(X)** de BASIC transfère la commande au sous-programme en langage machine situé à l'adresse stockée aux positions 785 et 786. (L'adresse est stockée en format standard octet de poids faible/octet de poids fort.) La valeur X est évaluée et passée au sous-programme en langage machine par l'accumulateur n° 1 à point flottant, placé à partir de l'adresse $61 (pour plus de détails, voir la topographie de mémoire). Une valeur peut se trouver retournée au programme BASIC en la plaçant dans l'accumulateur à point flottant. Le programme de routine en langage machine doit se terminer par une instruction RTS pour revenir au BASIC.

   Cette instruction diffère de **SYS**, car on doit établir un vecteur indirect. Le format par lequel passe la variable est également différent (format à point flottant). On doit changer le vecteur indirect si l'on utilise plus d'un programme en langage machine.

3) On peut remplacer ou modifier par un code d'utilisateur les programmes de routine internes BASIC ou d'entrée/sortie accessibles par la table de vecteurs située à la page 3 (voir **MODES D'ADRESSAGE, PAGE ZÉRO**). Chaque vecteur à 2 octets se compose d'une adresse d'octet de poids faible et d'octet de poids fort utilisée par le système d'exploitation.

   Le programme de routine de **vecteur KERNAL** est le moyen le plus sûr de changer un des vecteurs; on peut toutefois changer un seul vecteur par des instructions POKE. Un nouveau vecteur pointe à un programme de routine préparé par l'utilisateur qui doit remplacer ou compléter le programme de routine de système standard. À l'exécution de la commande BASIC correspondante, le programme d'utilisateur est exécuté. Si, après le passage du programme d'utilisateur, il faut exécuter le programme de système normal, le programme d'utilisateur doit sauter **(JMP)** à l'adresse précédemment contenue dans le vecteur. Si ce n'est pas le cas, le programme doit se terminer par une instruction RTS pour ramener la commande au BASIC.

4) On peut changer le **vecteur d'interrupteur de matériel (IRQ).** Chaque 1/60 de seconde, le système d'exploitation transfère la commande au programme spécifié par ce vecteur. Le KERNAL utilise normalement cette opération pour la synchronisation, l'exploration du clavier, etc. Avec cette technique, on doit toujours transférer la commande au programme de manipulation **IRQ** normal, sauf si le programme de remplacement est préparé pour la microplaquette CIA. (Ne pas oublier de terminer le programme de routine avec une instruction **RTI** (retour d'interruption) si la microplaquette CIA est traitée par ce programme.)

   Cette méthode est pratique avec les tâches qui doivent se produire parallèlement à un programme BASIC, mais elle présente l'inconvénient d'être plus difficile.

REMARQUE: Toujours invalider les interruptions avant de changer ce vecteur.

5) Le programme de routine **CHRGET** s'utilise en BASIC pour extraire chaque caractère/symbole. Il simplifie l'addition de nouvelles commandes BASIC. On doit naturellement exécuter chaque nouvelle commande avec un sous-programme en langage machine écrit par l'utilisateur. On peut normalement utiliser cette méthode en spécifiant un caractère (@ par exemple), qui doit se trouver avant toute nouvelle commande. Le nouveau programme de routine CHRGET recherche le caractère spécial. S'il n'en trouve pas, la commande passe au programme de routine CHRGET normal de BASIC. Si le caractère spécial est présent, le programme en langage machine interprète et exécute la nouvelle commande. On réduit ainsi le temps d'exécution supplémentaire ajouté par la nécessité de recherche d'autres commandes. Cette commande est dite *"insertion"*.

EMPLACEMENT DES PROGRAMMES DE ROUTINE EN LANGAGE MACHINE

Dans le Commodore 64, les programmes de routine en langage machine se placent de préférence de $C000 à $CFFF, à condition qu'ils aient moins de 4 K-octets de long. Cette section de la mémoire n'est pas perturbée par le BASIC.

Si, pour une raison quelconque, il n'est ni possible ni souhaitable de mettre le programme de routine en langage machine à $C000 (par exemple, si le programme a plus de 4 K-octets), il faut alors lui réserver une zone en haut de mémoire, à partir du BASIC. Le haut de mémoire est normalement à $9FFF. On peut changer le haut de mémoire par le programme de routine KERNAL **MEMTOP** ou en suivant les instructions BASIC suivantes:

```
10 POKE51,L:POKE52,H:POKE55,L:POKE56,H:CLR
```

Dans lesquelles H et L sont respectivement les parties haute et basse du nouveau haut de mémoire. Par exemple, pour réserver la zone de $9000 à $9FFF pour le langage machine, utiliser les instructions suivantes:

```
10 POKE51,0:POKE52,144:POKE55,0:POKE56,144:CLR
```

INTRODUCTION DU LANGAGE MACHINE

Il existe 3 méthodes courantes d'addition des programmes en langage machine à un programme BASIC. Ce sont:

1) INSTRUCTIONS DATA:

En lisant (READ) les instructions DATA et en inscrivant (POKE) les valeurs en mémoire au début du programme, on peut ajouter des programmes de routine en langage machine. Cette méthode est la plus facile. Il n'y a besoin d'aucune méthode spéciale pour sauvegarder les deux parties du programme et la mise au point est assez facile. Parmi les inconvénients, citons la plus grande place de mémoire nécessaire et l'attente pendant l'inscription (POKE) dans le programme. De ce fait, cette méthode est préférable avec les programmes de routine plus petits.

EXEMPLE:

```
10 RESTORE:FORX=1TO9:READA:POKE12*4096+X,A:NEXT
.
.
.
PROGRAMME BASIC
.
.
.
1000 DATA 161,1,204,204,204,204,204,204,96
```

2) CONTRÔLEUR DE LANGAGE MACHINE (64MON):

Ce programme permet d'introduire un programme en code hexadécimal ou symbolique et de sauvegarder la partie de mémoire où se trouve le programme. Cette méthode présente l'avantage de faciliter l'introduction des programmes de routine en langage machine, des auxiliaires de mise au point et d'accélérer la sauvegarde et le chargement. D'autre part, elle oblige le programme BASIC à charger le programme de routine en langage machine depuis la cassette ou le disque, à la mise en marche. (Pour plus de détails sur le 64MON, voir la section sur le langage machine.)

EXEMPLE:

Nous donnons ci-dessous un exemple de programme BASIC utilisant un programme de routine en langage machine préparé par le 64MON. Le programme de routine est stocké sur cassette:

```
10 IF FLAG=1 THEN 20
15 FLAG=1:LOAD "NOM DU PROGRAMME DE ROUTINE EN LANGAGE
MACHINE",1,1
20
.
.
.
RESTE DU PROGRAMME BASIC
```

3) PROGICIEL D'ÉDITEUR/ASSEMBLEUR:

Les avantages sont analogues à l'utilisation d'un contrôleur de langage machine, mais les programmes sont plus faciles à introduire. Les inconvénients sont également similaires à l'utilisation d'un contrôleur de langage machine.

TOPOGRAPHIE DE MÉMOIRE DU COMMODORE 64

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
D6510	0000	0	Registre de direction de données sur microplaquette 6510
R6510	0001	1	Registre d'entrée/sortie 8 bits sur microplaquette 6510
	0002	2	Non utilisé
ADRAY1	0003—0004	3—4	Vecteur de saut: conversion point flottant/entier
ADRAY2	0005—0006	5—6	Vecteur de saut: conversion entier/ point flottant
CHARAC	0007	7	Caractère de recherche
ENDCHR	0008	8	Indicateur: exploration pour guillemet en fin de chaîne
TRMPOS	0009	9	Colonne d'écran de dernière tabulation
VERCK	000A	10	Indicateur: 0 = charge, 1 = vérification
COUNT	000B	11	Pointeur de tampon d'entrée/nbre d'indices inférieurs
DIMFLG	000C	12	Indicateur: dimension de tableau implicite
VALTYP	000D	13	Type de données: $FF = chaîne, $00= numérique
INTFLG	000E	14	Type de données: $80 = entier, $00 = point flottant
GARBFL	000F	15	Indicateur: recherche de données/ guillemet LIST/regroupement positions inutilisées
SUBFLG	0010	16	Indicateur: référence indice inférieur/appel fonction utilisateur
INPFLG	0011	17	Indicateur: $00 = ENTRÉE, $40 =EXTRACTION, $98 = LECTURE
TANSGN	0012	18	Indicateur: signe tangente/résultat de comparaison
	0013	19	Indicateur: message-guide d'entrée

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
LINNUM	0014—0015	20—21	Temporaire: valeur entière
TEMPPT	0016	22	Pointeur: pile de chaîne temporaire
LASTPT	0017—0018	23—24	Dernière adresse de chaîne temporaire
TEMPST	0019—0021	25—33	Pile pour chaînes temporaires
INDEX	0022—0025	34—37	Zone de pointeur de programme utilitaire
RESHO	0026—002A	38—42	Produit de multiplication à point flottant
TXTTAB	002B—002C	43—44	Début du texte BASIC
VARTAB	002D—002E	45—46	Pointeur: début des variables BASIC
ARYTAB	002F—0030	47—48	Pointeur: début des tableaux BASIC
STREND	0031—0032	49—50	Pointeur: fin des tableaux BASIC (+1)
FRETOP	0033—0034	51—52	Pointeur: bas du stockage des chaînes
FRESPC	0035—0036	53—54	Pointeur de chaîne utilitaire
MEMSIZ	0037—0038	55—56	Pointeur d'adresse la plus haute utilisée par le BASIC
CURLIN	0039—003A	57—58	Numéro courant de ligne BASIC
OLDLIN	003B—003C	59—60	Numéro précédent de ligne BASIC
OLDTXT	003D—003E	61—62	Pointeur: instruction BASIC pour CONT
DATLIN	003F—0040	63—64	Numéro courant de ligne DATA
DATPTR	0041—0042	65—66	Pointeur: adresse courante d'article DATA
INPPTR	0043—0044	67—68	Vecteur: programme de routine INPUT
VARNAM	0045—0046	69—70	Nom courant de variable BASIC
VARPNT	0047—0048	71—72	Pointeur: données courantes de variable BASIC
FORPNT	0049—004A	73—74	Pointeur: variable d'index pour FOR/NEXT
	004B—0060	75—96	Pointeur temporaire/zone de données
FACEXP	0061	97	Accumulateur à point flottant n° 1: exposant
FACHO	0062—0065	98—101	Accumulateur flottant n° 1: mantisse
FACSGN	0066	102	Accumulateur à point flottant n° 1: signe
SGNFLG	0067	103	Pointeur: constante d'évaluation série

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
BITS	0068	104	Accumulateur à point flottant n° 1: chiffre de dépassement de capacité
ARGEXP	0069	105	Accumulateur à point flottant n° 2: exposant
ARGHO	006A—006D	106—109	Accumulateur à point flottant n° 2: mantisse
ARGSGN	006E	110	Accumulateur à point flottant n° 2: signe
ARISGN	006F	111	Résultat de comparaison de signe: accumulateurs n° 1 et n° 2
FACOV	0070	112	Accumulateur à point flottant n° 1; position de droite (arrondi)
FBUFPT	0071—0072	113—114	Pointeur: tampon de cassette
CHRGET	0073—008A	115—138	Sous-programme: extrait octet suivant de texte BASIC
CHRGOT	0079	121	Introduction pour extraction du même octet de texte
TXTPTR	007A—007B	122—123	Pointeur: octet courant de texte BASIC
RNDX	008B—008F	139—143	Valeur de base de fonction RND flottante
STATUS	0090	144	Mot d'état d'entrée/sortie de Kernal: ST
STKEY	0091	145	Indicateur: touche STOP/touche RVS
SVXT	0092	146	Constante de synchronisation pour bande
VERCK	0093	147	Indicateur: 0 = charge, 1 = vérification
C3PO	0094	148	Indicateur: bus série — caractère de sortie tamponné
BSOUR	0095	149	Caractère tamponné pour bus série
SYNO	0096	150	Numéro de synchro de cassette
	0097	151	Zone de données temporaire
LDTND	0098	152	Nbre de fichiers ouverts/index de table de fichier
DFLTN	0099	153	Dispositif d'entrée implicite (0)
DFLTO	009A	154	Dispositif de sortie (CMD) implicite (3)

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
PRTY	009B	155	Parité de caractère de cassette
DPSW	009C	156	Indicateur: octet de cassette — reçu
MSGFLG	009D	157	Indicateur: $80 = mode direct, $00 = programme
PTR1	009E	158	Journal d'erreur de passage 1 de cassette
PTR2	009F	159	Journal d'erreur de passage 2 de cassette
TIME	00A0—00A2	160—162	1/60 s (environ) d'horloge à temps réel
	00A3—00A4	163—164	Zone de données temporaire
CNTDN	00A5	165	Compte à rebours de synchro cassette
BUFPNT	00A6	166	Pointeur: tampon entrée/sortie cassette
INBIT	00A7	167	Bit d'entrée RS-232/temporaire cassette
BITCI	00A8	168	Compte de bits d'entrée RS-232/ temporaire cassette
RINONE	00A9	169	Indicateur RS-232: vérification du bit de départ
RIDATA	00AA	170	Tampon d'octet d'entrée RS-232/ temporaire cassette
RIPRTY	00AB	171	Parité d'entrée RS-232/compte court de cassette
SAL	00AC—00AD	172—173	Pointeur: tampon de bande/ défilement d'écran
EAL	00AE—00AF	174—175	Adresses de fin de bande/fin de programme
CMP0	00B0—00B1	176—177	Constantes de minutage de bande
TAPE1	00B2—00B3	178—179	Pointeur: début de tampon de bande
BITTS	00B4	180	Compte de bits de fin RS-232/ temporaire cassette
NXTBIT	00B5	181	Bit suivant RS-232 à envoyer/ indicateur fin de bande
RODATA	00B6	182	Tampon d'octet de fin RS-232
FNLEN	00B7	183	Longueur du nom de fichier courant
LA	00B8	184	Numéro du fichier logique courant

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
SA	00B9	185	Adresse secondaire courante
FA	00BA	186	Numéro de dispositif courant
FNADR	00BB—00BC	187—188	Pointeur: nom de fichier courant
ROPRTY	00BD	189	Parité de fin RS-232/temporaire cassette
FSBLK	00BE	190	Compteur de bloc lecture/écriture cassette
MYCH	00BF	191	Tampon de mot série
CAS1	00C0	192	Verrouillage moteur de magnétocassette
STAL	00C1—00C2	193—194	Adresse de départ entrée/sortie
MEMUSS	00C3—00C4	195—196	Temporaires de charge de bande
LSTX	00C5	197	Touche courante pressée: CHR$(n) 0= pas de touche
NDX	00C6	198	Nbre de caractères en tampon de clavier (file d'attente)
RVS	00C7	199	Indicateur: impression de caractères inverses −1 = oui, 0 = non utilisé
INDX	00C8	200	Pointeur: fin de ligne logique pour INPUT
LXSP	00C9—00CA	201—202	Position X-Y du curseur au début de INPUT
SFDX	00CB	203	Indicateur: imprime caractères avec SHIFT
BLNSW	00CC	204	Validation clignotement curseur: 0 = curseur clignotant
BLNCT	00CD	205	Minuterie: compte à rebours de basculement curseur
GDBLN	00CE	206	Caractère sous curseur
BLNON	00CF	207	Indicateur: dernier marche/arrêt de clignotement de curseur
CRSW	00D0	208	Indicateur: INPUT ou GET à partir du clavier
PNT	00D1-00D2	209—210	Pointeur: adresse courante de ligne d'écran

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
PNTR	00D3	211	Colonne du curseur sur ligne courante
QTSW	00D4	212	Indicateur: éditeur en mode guillemets, $00 = NON
LNMX	00D5	213	Longueur physique de ligne d'écran
TBLX	00D6	214	Numéro courant de ligne physique de curseur
	00D7	215	Zone de données temporaire
INSRT	00D8	216	Indicateur: mode d'insertion, >0 = nombre d'insertions
LDTB1	00D9—00F2	217—242	Table de lignes d'écran/temporaires d'éditeur
USER	00F3—00F4	243—244	Pointeur: position courante de mémoire vive de couleur d'écran
KEYTAB	00F5—00F6	245—246	Vecteur: table de décodage clavier
RIBUF	00F7—00F8	247—248	Pointeur de tampon d'entrée RS-232
ROBUF	00F9—00FA	249—250	Pointeur de tampon de sortie RS-232
FREKZP	00FB—00FE	251—254	Espace libre de page 0 pour programmes d'utilisateur
BASZPT	00FF	255	Zone de données temporaire BASIC
	0100-01FF	256—511	Zone de pile de système de microprocesseur
	0100-010A	256—266	Zone de travail point flottant/chaîne
BAD	0100—013E	256—318	Journal d'erreurs d'entrée bande
BUF	0200—0258	512—600	Tampon d'entrée du système
LAT	0259—0262	601—610	Table KERNAL: n° de fichiers logiques actifs
FAT	0263—026C	611—620	Table KERNAL: n° de dispositif pour chaque fichier
SAT	026D—0276	621—630	Table KERNAL: deuxième adresse, chaque fichier
KEYD	0277—0280	631—640	File d'attente de tampon de clavier (premier entré, premier sorti)

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
MEMSTR	0281—0282	641—642	Pointeur: bas de mémoire pour système d'exploitation
MEMSIZ	0283—0284	643—644	Pointeur: haut de mémoire pour système d'exploitation
TIMOUT	0285	645	Indicateur: variable Kernal pour délai d'attente IEEE
COLOR	0286	646	Code courant des couleurs de caractère
GDCOL	0287	647	Couleur de fond sous curseur
HIBASE	0288	648	Haut de mémoire d'écran (page)
XMAX	0289	649	Taille du tampon de clavier
RPTFLG	028A	650	Indicateur: touche REPEAT utilisée, $80 = répétition
KOUNT	028B	651	Compteur de vitesse de répétition
DELAY	028C	652	Compteur de retard de répétition
SHFLAG	028D	653	Indicateur: touche SHIFT du clavier/ touche CTRL/C= touche
LSTSHF	028E	654	Dernière configuration de décalage du clavier
KEYLOG	028F—0290	655—656	Vecteur: disposition de table du clavier
MODE	0291	657	Indicateur: $00=invalidation des touches SHIFT, $80= validation des touches SHIFT
AUTODN	0292	658	Indicateur: défilement automatique vers le bas, 0 = en fonction
M51CTR	0293	659	RS-232: image de registre de commande 6551
M51CDR	0294	660	RS-232: image de registre de commande 6551
M51AJB	0295—0296	661—662	Bauds non standard RS-232 (Temps/2-100) — É.U.
RSSTAT	0297	663	RS-232: image de registre d'état 6551
BITNUM	0298	664	RS-232: nombre de bits restant à envoyer

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
BAUDOF	0299—029A	665—666	Régime de bauds RS-232: durée de bit intégrale (μs)
RIDBE	029B	667	Index RS-232 vers tampon de fin d'entrée
RIDBS	029C	668	Début RS-232 de tampon d'entrée (page)
RODBS	029D	669	Début RS-232 de tampon de sortie (page)
RODBE	029E	670	Index RS-232 vers tampon de fin de sortie
IRQTMP	029F—02A0	671—672	Maintien du vecteur IRQ pendant l'entrée/sortie cassette
ENABL	02A1	673	Validation RS-232
	02A2	674	Détection de l'heure pendant entrée/sortie cassette
	02A3	675	Stockage temporaire pour lecture cassette
	02A4	676	Indicateur D1IRQ temporaire pour lecture cassette
	02A5	677	Temporaire pour index de ligne
	02A6	678	Indicateur PAL/NTSC, 0=NTSC, 1=PAL
	02A7—02FF	679—767	Non utilisé
IERROR	0300—0301	768—769	Vecteur: impression de message d'erreur BASIC
IMAIN	0302—0303	770—771	Vecteur: départ à chaud de BASIC
ICRNCH	0304—0305	772—773	Vecteur: symbolisation de texte BASIC
IQPLOP	0306—0307	774—775	Vecteur: liste de texte BASIC
IGONE	0308—0309	776—777	Vecteur: envoi de caractères BASIC
IEVAL	030A—030B	778—779	Vecteur: évaluation de symboles BASIC
SAREG	030C	780	Stockage pour registre .A de 6502
SXREG	030D	781	Stockage pour registre .X de 6502

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
SYREG	030E	782	Stockage pour registre .Y de 6502
SPREG	030F	783	Stockage pour registre .SP de 6502
USRPOK	0310	784	Instr. de saut de fonction USR (4C)
USRADD	0311—0312	785—786	Octet poids faible/octet poids fort
	0313	787	Non utilisé
CINV	0314—0315	788—789	Vecteur: interruption IRQ de matériel
CBINV	0316—0317	790—791	Vecteur: interruption d'instruction BRK
NMINV	0318—0319	792—793	Vecteur: interruption non invalidable
IOPEN	031A—031B	794—795	Vecteur de programme KERNAL OPEN
ICLOSE	031C—031D	796—797	Vecteur de programme KERNAL CLOSE
ICHKIN	031E—031F	798—799	Vecteur de programme KERNAL CHKIN
ICKOUT	0320—0321	800—801	Vecteur de programme KERNAL CHKOUT
ICLRCH	0322—0323	802—803	Vecteur de programme KERNAL CLRCHN
IBASIN	0324—0325	804—805	Vecteur de programme KERNAL CHRIN
IBSOUT	0326—0327	806—807	Vecteur de programme KERNAL CHROUT
ISTOP	0328—0329	808—809	Vecteur de programme KERNAL STOP
IGETIN	032A—032B	810—811	Vecteur de programme KERNAL GETIN
ICLALL	032C—032D	812—813	Vecteur de programme KERNAL CLALL
USRCMD	032E—032F	814—815	Vecteur défini par l'utilisateur
ILOAD	0330—0331	816—817	Vecteur de programme KERNAL LOAD
ISAVE	0332—0333	818—819	Vecteur de programme KERNAL SAVE
	0334—033B	820—827	Non utilisé
TBUFFR	033C—03FB	828—1019	Tampon d'entrée/sortie bande
	03FC—03FF	1020—1023	Non utilisé

INDICATIF	ADRESSE HEXADÉCIMALE	POSITION DÉCIMALE	DESCRIPTION
VICSCN	0400—07FF	1024—2047	Zone de mémoire d'écran de 1024 octets
	0400—07E7	1024—2023	Matrice vidéo: 25 lignes x 40 colonnes
	07F8—07FF	2040—2047	Pointeurs de données de caractère graphique programmable
	0800—9FFF	2048—40959	Espace de programme de BASIC normal
	8000—9FFF	32768—40959	Mémoire morte de cartouche VSP — 8192 octets
	A000—BFFF	40960—49151	Mémoire morte de BASIC — 8192 octets (ou 8 K de mémoire vive)
	C000—CFFF	49152—53247	Mémoire vive — 4096 octets
	D000—DFFF	53248—57343	Dispositifs d'entrée/sortie et mémoire RAM de couleur ou mémoire morte de générateur de caractère ou mémoire vive — 4096 octets
	E000—FFFF	57344—65535	Mémoire morte de KERNAL — 8192 octets (ou 8 K de mémoire vive)

AFFECTATIONS D'ENTRÉE/SORTIE DU COMMODORE 64

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
0000	0	7—0	Registre de direction de données de 6510 à MOS (xx101111) Bit=1: sortie, Bit=0: entrée, x=indifférent
0001	1		Accès d'entrée/sortie microplaquette de microprocesseur 6510 à MOS
		0	/signal mémoire vive (RAM) basse (0=coupure mémoire morte ROM de BASIC)

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		1	/Signal mémoire vive (RAM) haute (0 = coupure mémoire morte (ROM) de KERNAL)
		2	/Signal CHAREN (0 = mise en fonction mémoire morte de caractères)
		3	Ligne de sortie de données cassette
		4	Détection de commutation cassette 1 = interrupteur fermé
		5	Commande de moteur de cassette 0 = marche, 1 = arrêt
		6—7	Non défini
D000—D02E	53248—54271		CONTRÔLEUR D'INTERFACE VIDÉO 6566 À MOS (VIC)
D000	53248		Pos. X car. graph. progr. 0
D001	53249		Pos. Y car. graph. progr. 0
D002	53250		Pos. X car. graph. progr. 1
D003	53251		Pos. Y car. graph. progr. 1
D004	53252		Pos. X car. graph. progr. 2
D005	53253		Pos. Y car. graph. progr. 2
D006	53254		Pos. X car. graph. progr. 3
D007	53255		Pos. Y car. graph. progr. 3
D008	53256		Pos. X car. graph. progr. 4
D009	53257		Pos. Y car. graph. progr. 4
D00A	53258		Pos. X car. graph. progr. 5
D00B	53259		Pos. Y car. graph. progr. 5
D00C	53260		Pos. X car. graph. progr. 6
D00D	53261		Pos. Y car. graph. progr. 6
D00E	53262		Pos. X car. graph. progr. 7
D00F	53263		Pos. Y car. graph. progr. 7
D010	53264		Pos. X, car. graph. progr. 0—7 (bit le plus significatif de la coordonnée X)
D011	53265		Registre de commande VIC
		7	Comparaison de trame: (Bit 8) — Voir 53266
		6	Mode de texte couleur étendu: 1 = validation
		5	Mode de topographie de bits: 1 = validation

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		4	Passage de l'écran à la couleur du cadre: 0 = passage
		3	Choix d'affichage de texte 24/25 rangées: **1** = 25 rangées
		2—0	Défilement uniforme à la position de point Y (0—7)
D012	53266		Valeur de lecture trame/écriture trame pour comparaison IRQ
D013	53267		Position X de bascule de crayon lumineux
D014	53268		Position Y de bascule de crayon lumineux
D015	53269		Validation affichage de caractère graphique prog.: 1 = validation
D016	53270		Registre de commande VIC
		7—6	Non utilisé
		5	TOUJOURS METTRE CE BIT À ZÉRO!
		4	Mode multicolore: 1 = validation (texte ou topographie de bits)
		3	Choix de l'affichage de texte à 38/40 colonnes: 1 = 40 colonnes
		2—0	Défilement uniforme à la position X
D017	53271		Agrandissement des caractères graphiques programmables 0—7, 2 fois dans le sens vertical (Y)
D018	53272		Registre de commande de mémoire VIC
		7—4	Adresse de base de matrice vidéo (dans le VIC)
		3—1	Adresse de base de données de points de caractère (dans le VIC)
D019	53273		Registre d'indicateur d'interruption VIC (bit = 1: IRQ s'est produite)
		7	Réglage sur toute condition IRQ validée de VIC
		3	Indicateur IRQ déclenché de crayon lumineux

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		2	Indicateur IRQ de collision entre caractères graphiques programmables
		1	Indicateur IRQ de collision de caractère graphique programmable/ arrière-plan
		0	Indicateur IRQ de comparaison de trames
D01A	53274		Registre de masque IRQ: 1 = validation interruption
D01B	53275		Priorité d'affichage caractère graphique programmable/arrière-plan: 1 = caractère graphique programmable
D01C	53276		Sélection de mode multicolore, caractères graphiques programmables 0—7: 1 = mode multicolore
D01D	53277		Agrandissement caractères graphiques programmables 0—7, deux fois dans le sens horizontal (X)
D01E	53278		Détection de collision entre caractères graphiques programmables
D01F	53279		Détection de collision caractère graphique programmable/arrière-plan
D020	53280		Couleur de cadre
D021	53281		Couleur 0 d'arrière-plan
D022	53282		Couleur 1 d'arrière-plan
D023	53283		Couleur 2 d'arrière-plan
D024	53284		Couleur 3 d'arrière-plan
D025	53285		Registre multicolore 0 de caractère graphique programmable
D026	53286		Registre multicolore 1 de caractère graphique programmable
D027	53287		Couleur de caractère graphique programmable 0
D028	53288		Couleur de caractère graphique programmable 1
D029	53289		Couleur de caractère graphique programmable 2

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
D02A	53290		Couleur de caractère graphique programmable 3
D02B	53291		Couleur de caractère graphique programmable 4
D02C	53292		Couleur de caractère graphique programmable 5
D02D	53293		Couleur de caractère graphique programmable 6
D02E	53294		Couleur de caractère graphique programmable 7
D400--D7FF	54272—55295		DISPOSITIF D'INTERFACE DE SON 6581 À MOS (SID)
D400	54272		Voix 1: commande de fréquence — octet de poids faible
D401	54273		Voix 1: commande de fréquence — octet de poids fort
D402	54274		Voix 1: largeur de forme d'onde d'impulsion — octet de poids faible
D403	54275	7—4	Non utilisé
		3—0	Voix 1: largeur de forme d'onde d'impulsion — demi-octet de poids fort
D404	54276		Voix 1: registre de commande
		7	Sélection de forme d'onde de bruit aléatoire, 1 = marche
		6	Sélection de forme d'onde d'impulsion, 1 = marche
		5	Sélection de forme d'onde en dents de scie, 1 = marche
		4	Sélection de forme d'onde triangulaire, 1 = marche
		3	Bit de contrôle: 1 = invalidation oscillateur 1
		2	Oscillateur 1 de modulation directionnelle avec sortie oscillateur 3, 1 = marche
		1	Oscillateur 1 de synchronisation avec fréquence d'oscillateur 3, 1 = marche
		0	Bit de porte: 1 = début attaque/décroissance/stabilisation, 0 = début extinction

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
D405	54277		Générateur d'enveloppe 1: commande de cycle d'attaque/décroissance
		7—4	Sélection de durée de cycle d'attaque: 0-15
		3—0	Sélection de durée de cycle de décroissance: 0-15
D406	54278		Générateur d'enveloppe 1: commande de cycle de stabilisation/ extinction
		7—4	Sélection de durée de cycle de stabilisation: 0-15
		3—0	Sélection de durée de cycle d'extinction: 0-15
D407	54279		Voix 2: commande de fréquence — octet de poids faible
D408	54280		Voix 2: commande de fréquence — octet de poids fort
D409	54281		Voix 2: largeur de forme d'onde d'impulsion — octet de poids faible
D40A	54282	7—4	Non utilisé
		3—0	Voix 2: largeur de forme d'onde d'impulsion — demi-octet de poids fort
D40B	54283		Voix 2: registre de commande
		7	Sélection de forme d'onde de bruit aléatoire, 1 = marche
		6	Sélection de forme d'onde d'impulsion, 1 = marche
		5	Sélection de forme d'onde en dents de scie, 1 = 0
		4	Sélection de forme d'onde triangulaire, 1 = marche
		3	Bit de contrôle: 1 = invalidation oscillateur 2
		2	Oscillateur 2 de modulation directionnelle avec sortie oscillateur 1, 1 = marche
		1	Oscillateur de synchronisation 2 avec fréquence d'oscillateur 1, 1 = marche

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		0	Bit de porte: 1 = début attaque/ décroissance/stabilisation, 0 = début extinction
D40C	54284		Générateur d'enveloppe 2: commande de cycle d'attaque/ décroissance
		7—4	Sélection de durée de cycle d'attaque: 0-15
		3—0	Sélection de durée de cycle de décroissance: 0-15
D40D	54285		Générateur d'enveloppe 2: commande de cycle de stabilisation/extinction
		7—4	Sélection de durée de cycle de stabilisation: 0-15
		3—0	Sélection de durée de cycle d'extension: 0-15
D40E	54286		Voix 3: commande de fréquence — octet de poids faible
D40F	54287		Voix 3: commande de fréquence — octet de poids fort
D410	54288		Voix 3: largeur de forme d'onde d'impulsion — octet de poids faible
D411	54289	7—4	Non utilisé
		3—0	Voix 3: largeur de forme d'onde d'impulsion — demi-octet de poids fort
D412	54290		Voix 3: Registre de commande
		7	Sélection de forme d'onde de bruit aléatoire, 1 = marche
		6	Sélection de forme d'onde d'impulsion, 1 = marche
		5	Sélection de forme d'onde en dents de scie, 1 = marche
		4	Sélection de forme d'onde triangulaire, 1 = marche
		3	Bit de contrôle: 1 = invalidation oscillateur 3
		2	Oscillateur 3 de modulation directionnelle avec sortie oscillateur 2, 1 = marche

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		1	Oscillateur de synchronisation 3 avec fréquence d'oscillateur 2, 1 = marche
		0	Bit de porte: 1 = début attaque/ décroissance/stabilisation, 0 = début extinction
D413	54291		Générateur d'enveloppe 3: commande de cycle d'attaque/ décroissance
		7—4	Sélection de durée de cycle d'attaque: 0-15
		3—0	Sélection de durée de décroissance: 0-15
D414	54292		Générateur d'enveloppe 3: commande de cycle de stabilisation/ extinction
		7—4	Sélection de durée de cycle de stabilisation: 0-15
		3—0	Sélection de durée de cycle d'extinction: 0-15
D415	54293		Fréquence de coupure de filtre: demi-octet de poids faible (bits 2-0)
D416	54294		Fréquence de coupure de filtre: octet de poids fort
D417	54295		Commande de résonance de filtre/ commande de sortie voix
		7—4	Sélection de résonance de filtre: 0-15
		3	Entrée filtre externe: 1 = oui, 0 = non
		2	Sortie filtre voix 3: 1 = oui, 0 = non
		1	Sortie filtre voix 2: 1 = oui, 0 = non
		0	Sortie filtre voix 1: 1 = oui, 0 = non
D418	54296		Sélection de volume et mode de filtre
		7	Sortie coupure de voix 3: 1 = arrêt, 0 = marche
		6	Sélection de mode de filtre passe-haut: 1 = marche
		5	Sélection de mode de filtre de bande passante: 1 = marche

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		4	Sélection de mode de filtre passe-bas: 1 = marche
		3—0	Sélection de volume de sortie: 0-15
D419	54297		Convertisseur analogique/numérique: palette de jeu 1 (0-255)
D41A	54298		Convertisseur analogique/numérique: palette de jeu 2 (0-255)
D41B	54299		Générateur de nombre aléatoire d'oscillateur 3
D41C	54230		Sortie de générateur d'enveloppe 3
D500—D7FF	54528—55295		IMAGES SID
D800—DBFF	55296—56319		Mémoire vive de couleurs (demi octets)
DC00—DCFF	56320—56575		Adaptateur d'interface complexe 6526 à MOS (CIA) n° 1
DC00	56320		Accès A de données (clavier, manettes de jeu, palettes, crayon lumineux)
		7—0	Écriture des valeurs de colonne de clavier pour exploration du clavier
		7—6	Lecture des palettes sur accès A/B (01 = accès A, 10 = accès B)
		4	Boutons de tir de manette A: 1 = feu
		3—2	Boutons de tir de palette
		3—0	Direction de manette de jeu A (0-15)
DC01	56321		Accès B de données (clavier, manettes de jeu, palettes) accès de jeu 1
		7—0	Lecture des valeurs de rangée du clavier pour exploration du clavier
		7	Minuterie B: sortie bascule/impulsion
		6	Minuterie A: sortie bascule/impulsion
		4	Bouton de tir de manette 1: 1 = feu
		3—2	Boutons de tir de palette
		3—0	Direction de manette de jeu 1
DC02	56322		Registre de direction de données— Accès A (56320)
DC03	56323		Registre de direction de données — Accès B (56321)
DC04	56324		Minuterie A: octet de poids faible

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
DC05	56325		Minuterie A: octet de poids fort
DC06	56326		Minuterie B: octet de poids faible
DC07	56327		Minuterie B: octet de poids fort
DC08	56328		Horloge de l'heure: 1/10 s
DC09	56329		Horloge de l'heure: secondes
DC0A	56330		Horloge de l'heure: minutes
DC0B	56331		Horloge de l'heure: heures + indicateur AM/PM (bit 7)
DC0C	56332		Tampon de données d'entrée/sortie série synchrone
DC0D	56333		Registre de commande d'interruption CIA (lecture IRQ/écriture du masque)
		7	Indicateur IRQ (1 = IRQ s'est produite)/indicateur établissement-effacement
		4	Indicateur 1 IRQ (lecture cassette/ entrée SRQ de bus série)
		3	Interruption d'accès série
		2	Interruption d'alarme d'horloge de bord
		1	Interruption de minuterie B
		0	Interruption de minuterie A
DC0E	56334		Registre A de commande CIA
		7	Fréquence d'horloge de l'heure: 1 = 50 Hz, 0 = 60 Hz
		6	Mode d'entrée/sortie d'accès série 1 = sortie, 0 = entrée
		5	Comptes de minuterie A: 1 = signaux CNT, 0 = horloge de système 02
		4	Chargement forcé, minuterie A: 1 = oui
		3	Mode de marche de minuterie: 1 = pas à pas, 0 = continu
		2	Mode de sortie de minuterie à PB6: 1 = bascule, 0 = impulsion
		1	Sortie de minuterie A sur PB6: 1 = oui, 0 = non
		0	Minuterie de départ/arrêt A: 1 = départ, 0 = arrêt

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
DC0F	56335		Registre B de commande CIA
		7	Établissement alarme/horloge d'heure: 1 = alarme, 0 = horloge
		6—5	Sélection de mode de minuterie B: 00 = compte impulsions d'horloge de système 02 01 = compte transitions CNT positives 10 = compte impulsions de dépassement de capacité négatif de minuterie A 11 = compte dépassements de capacité négatifs de minuterie A avec CNT positif
		4—0	Identique au registre A de commande CIA pour minuterie B
DD00—DDFF	56576—56831		Adaptateur d'interface complexe 6526 MOS (CIA) n° 2
DD00	56576		Accès de données A (bus série RS-232, commande de mémoire VIC)
		7	Entrée de données de bus série
		6	Entrée d'impulsions d'horloge de bus série
		5	Sortie de données de bus série
		4	Sortie d'impulsions d'horloge de bus série
		3	Sortie de signal ATN de bus série
		2	Sortie de données RS-232 (accès utilisateur)
		1—0	Sélection de bloc de mémoire de microplaquette VIC (implicite = 11)
DD01	56577		Accès de données B (accès utilisateur, RS-232)
		7	Jeu de données utilisateur/RS-232 prêt
		6	Utilisateur/RS-232 prêt à l'envoi
		5	Utilisateur
		4	Détection de porteuse utilisateur/ RS-232

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		3	Indicateur de bague utilisateur/RS-232
		2	Terminal de données utilisateur/RS-232 prêt
		1	Demande d'envoi utilisateur/RS-232
		0	Données reçues utilisateur/RS-232
DD02	56578		Registre de direction de données—accès A
DD03	56579		Registre de direction de données — accès B
DD04	56580		Minuterie A : octet de poids faible
DD05	56581		Minuterie A : octet de poids fort
DD06	56582		Minuterie B : octet de poids faible
DD07	56583		Minuterie B : octet de poids fort
DD08	56584		Horloge de l'heure: 1/10 s
DD09	56585		Horloge de l'heure: secondes
DD0A	56586		Horloge de l'heure: minutes
DD0B	56587		Horloge de l'heure: heures et indicateur AM/PM (bit 7)
DD0C	56588		Tampon de données d'entrée/sortie série synchrone
DD0D	56589		Registre de commande d'interruption CIA (lecture interruption non invalidable/écriture masque)
		7	Indicateur NMI (I = NMI s'est produit/indicateur établissement-effacement)
		4	Indicateur I NMI (entrée de données reçues d'utilisateur/RS-232)
		3	Interruption d'accès série
		1	Interruption de minuterie B
		0	Interruption de minuterie A
DD0E	56590		Registre A de commande CIA
		7	Fréquence d'horloge de l'heure: 1 = 50 Hz, 0 = 60 Hz

HEXADÉCIMAL	DÉCIMAL	BITS	DESCRIPTION
		6	Mode d'entrée/sortie d'accès série 1 = sortie, 0 = entrée
		5	Comptes de minuterie A: 1 = signaux CNT, 0 = horloge de système 02
		4	Chargement forcé de minuterie A: 1 = oui
		3	Mode de marche de minuterie A: 1 = pas à pas, 0 = continu
		2	Mode de sortie de minuterie A à PB6: 1 = bascule, 0 = impulsion
		1	Sortie de minuterie A sur PB6: 1 = oui, 0 = non
		0	Minuterie A de départ/arrêt: 1 = départ, 0 = arrêt
DD0F	56591		Registre B de commande CIA
		7	Établissement alarme/horloge de l'heure: 1 = alarme, 0 = horloge
		6—5	Sélection de mode de minuterie B: 00 = compte impulsions d'horloge de système 02 01 = compte transitions CNT positives 10 = compte impulsions de dépassement de capacité négatif de minuterie A 11 = compte dépassements de capacité négatifs de minuterie A avec CNT positif
		4—0	Identique au registre A de commande CIA— pour minuterie B
DE00—DEFF	56832—57087		Réservé pour extension d'entrée/ sortie future
DF00—DFFF	57088—57343		Réservé pour extension d'entrée/ sortie future

CHAPITRE 6

GUIDE D'ENTRÉE/SORTIE

- Introduction
- Sortie vers le télécouleur
- Sortie vers d'autres dispositifs
- Accès de jeux
- Description de l'interface RS-232
- Accès d'utilisateur
- Bus série
- Accès d'extension
- Cartouche de microprocesseur Z-80

INTRODUCTION

L'ordinateur remplit trois fonctions essentielles: il peut calculer, prendre des décisions et communiquer. Le calcul est probablement la fonction la plus facile à programmer. Nous connaissons la plupart des règles de mathématique. La prise de décision ne présente plus guère de difficultés, car les règles de logique sont relativement peu nombreuses, même si nous ne les connaissons pas encore trop bien.

La communication est plus complexe, car elle s'appuie sur des règles qui manquent de rigueur. Il ne s'agit pas d'une négligence dans la conception des ordinateurs. Les règles présentent une flexibilité suffisante pour communiquer une grande diversité d'éléments, sous de nombreuses formes possibles. La seule règle véritable veut que les informations soient présentées de façon à être comprises au point de réception.

SORTIE VERS LE TÉLÉCOULEUR

L'instruction PRINT représente la forme la plus simple de sortie en BASIC. PRINT utilise l'écran de visualisation comme dispositif de sortie. L'oeil de l'utilisateur correspond au dispositif d'entrée, car il lit l'information sur l'écran.

Quand on imprime (PRINT) sur l'écran, on cherche surtout à mettre l'information sous une forme facile à lire. On doit penser comme un graphiste et utiliser les couleurs, la disposition des lettres majuscules et minuscules ainsi que des éléments graphiques pour mieux communiquer l'information. Quelle que soit la qualité du programme, il faut se rappeler qu'on doit être en mesure d'en comprendre les résultats.

L'instruction PRINT utilise certains codes de caractères comme "commandes" du curseur. La touche CRSR n'affiche rien; elle ne fait que changer la position du curseur. D'autres commandes changent les couleurs, effacent l'écran et insèrent ou annulent les espaces. Le numéro de code de caractère (CHR$) de la touche RETURN est 13. L'annexe C donne une table complète de ces codes.

En langage BASIC, deux fonctions sont associées à l'instruction PRINT. TAB (tabulation) place le curseur à la position donnée à partir du bord gauche de l'écran. SPC (espace) déplace le curseur vers la droite d'un certain nombre d'espaces à partir de la position présente.

Dans l'instruction PRINT, les signes de ponctuation servent à séparer les informations et à les mettre en forme. Le point-virgule (;) sépare deux éléments, sans espace. Si le point-virgule est le dernier signe d'une ligne, le curseur ne passe pas à la ligne suivante après l'impression du dernier élément. Il remplace (supprime) le caractère RETURN normalement imprimé (PRINT) à la fin de la ligne.

La virgule (,) sépare les éléments en colonnes. Le Commodore 64 permet 4 colonnes de 10 caractères chacune sur l'écran. Quand l'ordinateur imprime (PRINT) une virgule, il déplace le curseur au début de la colonne suivante. Si le curseur est au-delà de la dernière colonne de la ligne, il passe à la ligne suivante. Comme pour le point-virgule, le caractère RETURN est supprimé si une virgule termine la ligne.

Les guillemets ("") séparent le texte littéral des variables. Le premier guillemet de la ligne commence la zone littérale et le guillemet suivant la termine. Signalons qu'il n'est pas utile d'avoir un guillemet final à la fin de la ligne.

Le code RETURN (CHR$ 13) fait passer le curseur à la ligne *logique* suivante de l'écran qui n'est pas toujours la ligne suivante proprement dite. Quand on continue à taper après la fin d'une ligne, celle-ci est liée à la ligne suivante. L'ordinateur sait que les deux lignes forment en réalité une ligne longue. Les liens sont gardés dans la table de *liens de lignes* (voir la topographie de mémoire pour cette disposition).

Une ligne logique peut avoir une ou deux lignes d'écran de long, suivant ce qui a été tapé ou imprimé (PRINT). La ligne logique où se trouve le curseur détermine où il est envoyé par la touche RETURN . La ligne logique en haut de l'écran détermine si celui-ci défile d'une ou deux lignes à la fois.

Il existe d'autres façons d'utiliser le télécouleur comme dispositif de sortie. Le chapitre relatif aux graphiques indique les commandes à utiliser pour créer des objets se déplaçant sur l'écran. La section sur la microplaquette VIC explique le changement des dimensions et des couleurs de l'écran et de son cadre. Le chapitre sur la sonorisation permet d'obtenir musique et effets spéciaux avec le haut-parleur du télécouleur.

SORTIE VERS D'AUTRES DISPOSITIFS

On doit souvent envoyer une sortie vers des dispositifs autres que l'écran (magnétocassette, imprimante, unité de disque ou modem). En BASIC, l'instruction OPEN crée un "canal" pour communiquer avec l'un de ces dispositifs. Quand le canal est ouvert, l'instruction PRINT# envoie des caractères à ce dispositif.

EXEMPLE d'instructions OPEN et PRINT#:

```
100 OPEN 4,4: PRINT# 4, "ECRITURE SUR L'IMPRIMANTE"
110 OPEN 3,8,3, "0:FICHIER-DISQUE,S,W": PRINT# 3, "ENVOI AU DISQUE"
120 OPEN 1,1,1, "FICHIER CASSETTE": PRINT# 1, "ECRITURE SUR BANDE"
130 OPEN 2,2,0, CHR$(10): PRINT# 2, "ENVOI AU MODEM"
```

L'instruction OPEN diffère pour chaque dispositif. La table ci-dessous donne les paramètres de l'instruction OPEN pour chaque dispositif.

TABLE des paramètres de l'instruction OPEN:

FORMAT: OPEN numéro de fichier, numéro de dispositif, numéro, chaîne

DISPOSITIF	NUMÉRO DISPOSITIF	NUMÉRO	CHAÎNE
CASSETTE	1	0 = entrée 1 = sortie 2 = sortie avec fin de bande	Nom de fichier
MODEM	2	0	Registres de commande
ÉCRAN	3	0,1	
IMPRIMANTE	4 ou 5	0 = majuscules/ graphiques 7 = majuscules/ minuscules	Impression (PRINT) du texte
DISK	8 à 11	2—14 = canal de données 15 = canal de commande	Numéro d'unité, nom de fichier, type de fichier, commande lecture/ écriture

SORTIE VERS IMPRIMANTE

L'imprimante est un dispositif de sortie analogue à l'écran. Quand on envoie une sortie vers l'imprimante, on doit surtout se préoccuper de créer un format agréable à l'oeil. Dans ce but, on dispose de la vidéo inverse, de la double largeur, des majuscules et des minuscules ainsi que des graphiques programmables par points.

Avec l'imprimante, la fonction SPC donne les mêmes résultats qu'avec l'écran. La fonction TAB ne donne cependant pas de bons résultats avec l'imprimante, car elle calcule la position présente de la ligne, d'après la position du curseur sur l'écran et non sur le papier.

L'instruction OPEN de l'imprimante crée le canal de communication. Elle spécifie aussi le jeu de caractères utilisé (majuscules avec graphiques ou majuscules et minuscules).

EXEMPLES d'instruction OPEN pour l'imprimante:

```
OPEN 1, 4: REM MAJUSCULES/GRAPHIQUES
OPEN 1, 4, 7: REM MAJUSCULES ET MINUSCULES
```

Si l'on travaille avec un jeu de caractères, on peut imprimer des lignes individuelles dans le jeu de caractères opposé. En majuscules avec graphiques, le caractère de descente du curseur (CHR$(17)) fait passer les caractères au jeu de majuscules et minuscules. En majuscules et minuscules, le caractère de montée de curseur (CHR$(145)) permet l'impression (PRINT) des caractères majuscules et graphiques.

Des codes de caractère commandent d'autres fonctions spéciales de l'imprimante. Ces codes s'impriment (PRINT) exactement comme tout autre caractère.

TABLE des codes de caractère de commande d'imprimante:

CODE CHR$	BUT
10	Avance de ligne
13	RETURN (avance automatique de ligne avec imprimantes CBM)
14	Commence le mode de caractères en double largeur
15	Termine le mode de caractères en double largeur
18	Commence le mode de caractères en vidéo inverse
146	Termine le mode de caractères en vidéo inverse
17	Passe au jeu de caractères majuscules/minuscules
145	Passe au jeu de caractères majuscules/graphiques
16	Tabulation à la position des 2 caractères suivants
27	Va à la position de point spécifiée
8	Commence le mode de graphique programmable par points
26	Répète les données de graphiques

Consulter le manuel d'imprimante Commodore pour plus de détails sur l'utilisation des codes de commande.

SORTIE VERS UN MODEM

Le modem est un dispositif simple capable de traduire les codes de caractère en impulsions sonores et vice versa pour que les ordinateurs puissent communiquer par les lignes téléphoniques. Pour le modem, l'instruction OPEN fixe les paramètres pour s'adapter à la vitesse et au format de l'ordinateur avec lequel on communique. On peut envoyer deux caractères dans la chaîne, à la fin de l'instruction OPEN.

Les positions de bits du premier code de caractère déterminent le régime de bauds, le nombre de bits de données et le nombre de bits d'arrêt. Le deuxième code est facultatif; ses bits précisent la parité et le type de duplex de la transmission. Pour plus de détails sur ce dispositif, consulter la section RS-232 du manuel **VICMODEM.**

EXEMPLE d'instruction OPEN pour un modem:

```
OPEN 1, 2, 0, CHR$(6): REM 300 BAUDS
100 OPEN 2, 2, 0, CHR$(163) CHR$(112): REM 110 BAUDS, ETC.
```

La plupart des ordinateurs emploient le code ASCII (code standard américain pour l'échange d'informations). Ce jeu standard de codes de caractère diffère notablement des codes utilisés dans le Commodore 64. Pour communiquer avec d'autres ordinateurs, on doit traduire les codes de caractère Commodore dans leurs équivalents ASCII. L'annexe C de ce manuel donne une table des codes ASCII standard.

La traduction des caractères mise à part, la sortie vers le modem est assez simple. On doit cependant bien connaître le dispositif récepteur, surtout si l'on écrit des programmes dans lesquels le Commodore 64 communique avec un autre ordinateur, sans intervention humaine. À titre d'exemple, un programme de terminal peut taper automatiquement un numéro de compte et un mot de passe secret. Pour accomplir cette tâche de façon satisfaisante, on doit compter attentivement le nombre de caractères et les retourner (RETURN), sinon l'ordinateur récepteur ne saura à quelle fin les utiliser.

UTILISATION DES CASSETTES

Les cassettes ont une capacité de données pratiquement illimitée. La capacité de stockage dépend de la longueur de la bande. Les cassettes sont cependant limitées dans le temps. La durée nécessaire pour localiser des informations dépend de la capacité de données de la cassette.

Le programmeur doit s'efforcer de minimiser le facteur temps quand il travaille avec des cassettes. En général, on lit le fichier de données de la cassette entière en mémoire vive, on le traite et on réécrit toutes les données sur bande. On peut trier, éditer et examiner les données. Cette pratique limite cependant la taille des fichiers à la quantité de mémoire vive disponible.

Si le fichier de données est plus grand que la mémoire vive disponible, il est alors recommandé d'adopter le disque souple. On peut lire certaines données sur le disque sans devoir filtrer toutes les autres. On peut écrire des données par-dessus les anciennes sans perturber le reste du fichier. Pour cette raison, on utilise le disque dans toutes les applications de gestion, comme le grand livre et les listes de publipostage.

L'instruction PRINT# met les données en forme, exactement comme l'instruction PRINT. Les signes de ponctuation remplissent les mêmes fonctions. Il faut cependant se rappeler que l'on ne travaille plus avec l'écran. On doit faire la mise en forme à l'aide de l'instruction PRINT#.

Envisageons l'instruction PRINT# 1, A$, B$, C$. Sur l'écran, les virgules qui séparent les variables donnent des espaces vides assez longs entre les éléments pour les disposer en colonnes de 10 caractères de large. Sur cassette, les virgules ajoutent de 1 à 10 espaces, suivant la longueur des chaînes. Cette disposition gaspille de la place de stockage.

La situation s'aggrave quand l'instruction INPUT# essaie de lire ces chaînes. L'instruction INPUT# 1, A$, B$, C$ ne trouve aucune donnée pour B$ et C$. A$ contient les trois variables ainsi que les espaces qui les séparent. Que s'est-il passé? Jetons un coup d'oeil au fichier de cassette:

```
A$="BOL" B$="TAS" C$="ZONE"
PRINT# 1, A$, B$, C$
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
B	O	L								T	A	S								Z	O	N	E	RETURN

L'instruction INPUT# marche comme l'instruction INPUT normale. Quand on tape des données dans l'instruction INPUT, les éléments sont séparés par la touche RETURN ou par des virgules. L'instruction PRINT# place RETURN à la fin d'une ligne, comme l'instruction PRINT. A$ se remplit des trois valeurs, car il y a un séparateur à leur suite sur la bande et non entre elles.

Sur la bande, on peut utiliser une virgule (,) ou un code RETURN comme séparation. Le code RETURN est automatiquement placé à la fin d'une instruction PRINT ou PRINT#. Pour mettre un code RETURN entre chaque élément, on n'utilise qu'un seul élément par instruction PRINT#. On peut fixer une variable au code CHR$ RETURN, qui correspond à CHR$(13), ou utiliser une virgule. L'instruction correspondante est R$=",":PRINT# 1,A$ R$ B$ R$ C$. Ne pas utiliser de virgule ou d'autres signes de ponctuation entre les noms de variables, car le Commodore 64 peut les distinguer et ils ne font qu'occuper de la place dans le programme.

Un fichier correct sur bande a l'aspect suivant:

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
B	**O**	**L**	**,**	**T**	**A**	**S**	**,**	**Z**	**O**	**N**	**E**	**RETURN**

L'instruction GET# extrait les données de la bande à raison d'un caractère à la fois. Elle reçoit chaque caractère, y compris le code RETURN et les signes de ponctuation. Le code CHR$(0) correspond à une chaîne vide et non à une chaîne de caractères portant le code 0. Si l'on essaie d'utiliser la fonction ASC avec une chaîne vide, on obtient le message d'erreur **ILLEGAL QUANTITY ERROR** (erreur de quantité interdite).

On utilise fréquemment la ligne GET# 1, A$: A = ASC(A$) dans les programmes pour examiner les données sur cassette. Pour éviter les messages d'erreur, modifier la ligne

pour avoir GET#1, A$: A= ASC(A$ + CHR$(0)). Le CHR$(0) final sert de garantie contre les chaînes vides, mais il n'affecte pas la fonction ASC en présence d'autres caractères dans A$.

STOCKAGE DES DONNÉES SUR MINIDISQUE SOUPLE

Les minidisques offrent trois formes différentes de stockage des données. Les fichiers séquentiels sont analogues à ceux sur cassette, mais on peut en utiliser plusieurs en même temps. Les fichiers relatifs permettent d'organiser des données en fiches, puis de lire et remplacer les fiches individuelles dans le fichier. Les fichiers aléatoires permettent de travailler avec les données en tout point du disque. Les fichiers sont disposés en sections de 256 octets ou blocs.

Nous avons étudié les limites de l'instruction PRINT# dans la section sur les cassettes. Les mêmes limites de mise en forme s'appliquent aux disques. On doit utiliser des codes RETURN ou des virgules pour séparer les données. CHR$(0) correspond encore à une chaîne vide pour l'instruction GET#.

Dans les fichiers relatifs et aléatoires, on utilise des canaux séparés de données et de commandes. Les données écrites sur les disques passent par le canal de données où elles sont stockées dans un tampon temporaire de la mémoire vive de disque. Quand la fiche ou bloc est plein, une commande, envoyée par le canal de commande, indique le point de placement des données à l'unité; le tampon entier est alors écrit.

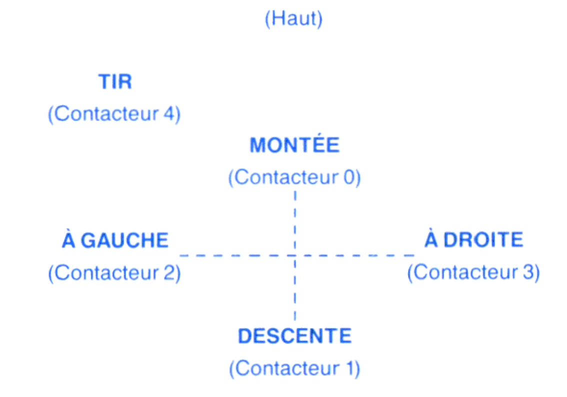
Il est préférable de stocker dans des fichiers relatifs sur disque les applications qui font appel à de grandes quantités de données à traiter. Ces fichiers demandent moins de temps et assurent une meilleure flexibilité pour le programmeur. Le manuel de l'unité de disque donne un guide complet de programmation pour l'utilisation des fichiers de disque.

ACCÈS DE JEUX

Le Commodore 64 est équipé de deux accès de jeux à 9 broches qui permettent l'utilisation des manettes de commande, des palettes ou d'un crayon lumineux. Chaque accès peut recevoir une manette de commande ou une paire de palettes. On peut brancher un crayon lumineux dans l'accès A seulement pour la commande graphique spéciale, etc. Cette section donne des exemples d'utilisation des manettes de commande et des palettes avec le BASIC et le langage machine.

La manette de commande numérique se raccorde à l'adapteur d'interface complexe 6526 à MOS (CIA nº 1). Ce dispositif d'entrée/sortie prend aussi en charge les boutons de tir de palette et l'exploration du clavier. La microplaquette CIA 6526 possède 16 registres dans les positions de mémoire 56320 à 56335 comprises ($DC00 à $DC0F). Les données de l'accès A apparaissent à la position 56320 (DC00) et les données de l'accès B à la position 56321 ($DC01).

Une manette de commande numérique comprend cinq contacteurs distincts dont quatre servent à la commande de direction et un au bouton de tir. Les contacteurs de la manette sont disposés de la façon suivante:



Ces contacteurs correspondent aux cinq bits inférieurs des données à la position 56320 ou 56321. Le bit est normalement fixé à 1 si l'on n'a pas choisi une direction ou si l'on n'a pas pressé le bouton de tir. Quand on appuie sur le bouton de tir, le bit 4 passe à 0. Pour lire la manette de commande à partir du BASIC, utiliser le sous-programme suivant:

```
10 FORK=0TO10:REM ETABLISSEMENT DE LA CHAINE DE DIRECTION
20 READDR$(K):NEXT
30 DATA"","N","S","","W","NW"
40 DATA"SW","","E","NE","SE"
50 PRINT"VERS . . . ";
60 GOSUB100:REM LECTURE DE LA MANETTE DE COMMANDE
65 IFDR$(JV)=""THEN80:REM VERIFIE SI UNE DIRECTION A ETE
CHOISIE
70 PRINTDR$(JV);" ";:REM SORTIE DE LA DIRECTION CHOISIE
80 IFFR=16THEN60:REM VERIFIE SI LE BOUTON DE TIR EST PRESSE
90 PRINT"------T------I------R------!!!":GOTO60
100 JV=PEEK(56320):REM EXTRAIT LA VALEUR DE LA MANETTE DE COMMANDE
110 FR=JVAND16:REM FORME L'ETAT DU BOUTON DE TIR
120 JV=15-(JVAND15):REM FORME LA VALEUR DE DIRECTION
130 RETURN
```

REMARQUE: Pour la deuxième manette de commande, taper JV = PEEK (56321).

Les valeurs de JV correspondent aux directions suivantes:

JV ÉGAL À	DIRECTION
0	AUCUNE
1	MONTEE
2	DESCENTE
3	—
4	À GAUCHE
5	MONTÉE VERS LA GAUCHE
6	DESCENTE VERS LA GAUCHE
7	—
8	À DROITE
9	MONTÉE VERS LA DROITE
10	DESCENTE VERS LA DROITE

Le petit programme de routine suivant en code machine donne les mêmes résultats:

```
1000 .PAGE (MANETTE DE COMMANDE.8/5) MANETTE DE COMMANDE — PRO-
GRAMME DE LECTURE DU BOUTON
1010 ;
1020 ;AUTEUR — BILL HINDORFF
1030 ;
1040 DX=$C110
1050 DY=$C111
1060 *=$C200
1070 DJRR        LDA $DC00        ;   (EXTRAIT DE L'ACCES A SEULEMENT)
1080 DJRRB       LDY #0           ;CE PROGRAMME DE ROUTINE LIT ET
DECODE LES
1090             LDX #0           ;DONNEES D'ENTREE DE MANETTE
DE COMMANDE/BOUTON DE TIR DANS
1100             LSR A            ;L'ACCUMULATEUR. LES CINQ BITS
LES MOINS
```

```
1110            BCS DJR0        ;SIGNIFICATIFS CONTIENNENT LES
INFORMATIONS DE
1120            DEY             ;FERMETURE DES CONTACTEURS. SI
UN CONTACTEUR EST FERME, IL
1130 DJR0       LSR A           ;DONNE ALORS UN BIT ZERO. SI
UN CONTACTEUR EST OUVERT, IL
1140            BCS DJR1        ;DONNE ALORS UN BIT UN.
LES DIRECTIONS DE LA MANETTE DE COMMANDE
1150            INY             ;VONT VERS LA DROITE ET LA GAUCHE
ET VERS L'AVANT ET L'ARRIERE
1160 DJR1       LSR A           ;BIT3=A DROITE, BIT2=A GAUCHE,
BIT1=VERS L'ARRIERE,
1170            BCS DJR2        ;BIT0 =VERS L'AVANT ET
BIT4=BOUTON DE TIR.
1180            DEX             ;AU MOMENT RTS DX ET DY
CONTIENNENT LE COMPLEMENT A 2
1190 DJR2       LSR A           ;DES NOMBRES DE DIRECTION,
C'EST-A-DIRE $FF=-1, $00 = 0, $01=1.
1200            BCS DJR3        ;DX=1 (DEPLACEMENT A DROITE), DX=-1
(DEPLACEMENT A GAUCHE),
1210            INX             ;DX= 0 (PAS DE CHANGEMENT X).
DY=-1 (MONTEE SUR L'ECRAN),
1220 DJR3       LSR A           ;DY=1 (DESCENTE SUR L'ECRAN),
DY= 0 (PAS DE CHANGEMENT Y).
1230            STX DX          ;LA POSITION VERS L'AVANT DE LA
MANETTE DE COMMANDE CORRESPOND
1240            STY DY          ;A LA MONTEE SUR L'ECRAN
ET LA POSITION
1250            RTS             ;VERS L'ARRIERE CORRESPOND A LA
DESCENTE SUR L'ECRAN.
1260 ;
1270 ;AU MOMENT RTS, L'INDICATEUR DE REPORT CONTIENT L'ETAT DU BOUTON
DE TIR.
1280 ;IF C=1 THEN BOUTON NON PRESSE. IF C= 0 THEN BOUTON PRESSE.
1290 ;
1300 .END
```

PALETTES

Une palette se raccorde aux microplaquettes CIA nº 1 et SID (dispositif d'interface de son 6581 à MOS) par un accès de jeu. Les registres SID 54297 ($D419) et 54298 ($D41A) lisent la valeur de palette. LES PALETTES MANQUENT DE FIABILITÉ SI ELLES SONT LUES DU BASIC SEULEMENT! Pour utiliser les palettes à partir du BASIC ou du langage machine, il est préférable d'utiliser le programme de routine suivant en langage machine (avec instruction SYS à partir du BASIC puis lecture (PEEK) des positions de mémoire utilisées par le sous-programme).

```
1000
;*********************************************************************************************
1010 ;* PROGRAMME DE LECTURE DE QUATRE PALETTES (PEUT AUSSI SERVIR
POUR DEUX)
1020
;*********************************************************************************************
1030 ;AUTEUR — BILL HINDORFF
1040 PORTA=$DC00
1050 CIDDRA=$DC02
1060 SID=$D400
1070 *=$C100
1080 BUFFER *=*+1
1090 PDLX *=*+2
1100 PDLY *=*+2
1110 BTNA *=*+1
1120 BTNB *=*+1
1130 *=$C000
1140 PDLRD
1150          LDX #1          ;POUR QUATRE PALETTES OU DEUX BATONS
DE COMMANDE ANALOGIQUES
1160 PDLRD0                   ;POINT D'ENTREE D'UNE PAIRE (CONDITION
X EN PREMIER)
1170          SEI
1180          LDA CIDDRA      ;EXTRAIT LA VALEUR COURANTE DE DDR
1190          STA BUFFER      ;LA SAUVEGARDE
1200          LDA #$C0
1210          STA CIDDRA      ;FIXE L'ACCES A POUR L'ENTREE
1220          LDA #$80
1230 PDLRD1
1240          STA PORTA       ;ADRESSE D'UNE PAIRE DE PALETTES
```

```
1250            LDY #$80        ;LEGERE ATTENTE
1260 PDLRD2
1270            NOP
1280            DEY
1290            BPL PDLRD2
1300            LDA SID+25      ;EXTRAIT LA VALEUR X
1310            STA PDLX,X
1320            LDA SID+26      ;EXTRAIT LA VALEUR Y
1330            STA PDLY,X
1340            LDA PORTA       ;LECTURE DES BOUTONS DE TIR DES PALETTES
1350            ORA #$80        ;MEME CHOSE QUE POUR L'AUTRE PAIRE
1360            STA BTNA        ;BIT 2 POUR PDL X, BIT 3 POUR PDL Y
1370            LDA #$40
1380            DEX             ;TOUTES LES PAIRES SONT-ELLES LUES?
1390            BPL PDLRD1      ;NON
1400            LDA BUFFER
1410            STA CIDDRA      ;RETABLIT LA VALEUR PRECEDENTE DE DDR
1420            LDA PORTA+1     ;POUR DEUXIÈME PAIRE —
1430            STA BTNB        ;LE BIT 2 EST PDL X, LE BIT 3 EST PDL Y
1440            CLI
1450            RTS
1460 .END
```

On peut lire les palettes à l'aide du programme BASIC suivant:

```
10 C=12*4096:REM FIXE LE DEBUT DU PROGRAMME DE PALETTE
11 REM ECRIT DANS LE PROGRAMME DE LECTURE DE PALETTE
15 FORI=0TO63:READA:POKEC+I,A:NEXT
20 SYSC:REM APPELLE LE PROGRAMME DE ROUTINE DE PALETTE
30 PI=PEEK(C+257):REM FIXE LA VALEUR DE LA PALETTE UN
40 P2=PEEK(C+258):REM FIXE LA VALEUR DE LA PALETTE DEUX
50 P3=PEEK(C+259):REM FIXE LA VALEUR DE LA PALETTE TROIS
60 P4=PEEK(C+260):REM FIXE LA VALEUR DE LA PALETTE QUATRE
61 REM LIT L'ETAT DES BOUTONS DE TIR
62 S1=PEEK(C+261):S2=PEEK(C+262)
70 PRINTP1,P2,P3,P4:REM IMPRIME LES VALEURS DES PALETTES
72 REM IMPRIME L'ETAT DES BOUTONS DE TIR
75 PRINT:PRINT"TIR A";S1,"TIR B";S2
```

```
80 FORW=1TO50:NEXT:REM LEGERE ATTENTE
            SHIFT CLR/HOME
90 PRINT "♥":PRINT:GOTO 20:REM EFFACE L'ECRAN ET RECOMMENCE
95 REM DONNEES POUR LE PROGRAMME DE ROUTINE EN LANGAGE MACHINE
100 DATA162,1,120,173,2,220,141,0,193,169,192,141,2,220,169
110 DATA128,141,0,220,160,128,234,136,16,252,173,25,212,157
120 DATA1,193,173,26,212,157,3,193,173,0,220,9,128,141,5,193
130 DATA169,64,202,16,222,173,0,193,141,2,220,173,1,220,141
140 DATA6,193,88,96
```

CRAYON LUMINEUX

L'entrée de crayon lumineux verrouille la position présente d'écran dans une paire de registres (LPX et LPY) sur front descendant d'impulsion. Le registre 19 de position X ($13) contient les 8 bits les plus significatifs de la position X au moment de la transition. La position X étant définie par un compteur à 512 états (9 bits), on obtient une définition de 2 points horizontaux. De même, la position Y est verrouillée dans son registre 20 ($14), mais 8 bits donnent ici une définition de trame simple sur l'affichage visible. On peut déclencher la bascule de crayon lumineux une fois seulement par trame, les déclenchements ultérieurs dans la même trame n'ont aucun effet. De ce fait, on doit prendre plusieurs échantillons avant de placer le crayon sur l'écran (3 ou plusieurs échantillons en moyenne), suivant les caractéristiques de l'ustensile.

DESCRIPTION DE L'INTERFACE RS-232

GÉNÉRALITÉS

Le Commodore 64 possède une interface RS-232 intégrée pour le raccordement à un modem, imprimante ou autre dispositif RS-232. Pour accorder un dispositif au Commodore 64, il suffit d'un câble et d'un peu de programmation.

Le RS-232 du Commodore 64 est établi en format RS-232 standard, mais les tensions sont aux niveaux TTL (0 à 5 V) plutôt que dans l'intervalle normal RS-232 de −12 à 12 volts. Le câble qui joint le Commodore 64 et le dispositif RS-232 doit assurer les conversions nécessaires de tension. La cartouche d'interface RS-232 de Commodore assure correctement cette fonction.

On peut accéder au logiciel d'interface RS-232 par le BASIC ou par le KERNAL pour la programmation en langage machine.

Au niveau du BASIC, le RS-232 emploie les commandes BASIC normales: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT# et la variable réservée ST. INPUT# et GET# extraient les données du tampon récepteur; PRINT# et CMD placent les données dans le tampon émetteur. Nous décrivons plus en détail l'utilisation de ces commandes (avec exemples) dans la suite de ce chapitre.

Les programmes de traitement KERNAL RS-232 au niveau de l'octet et du bit sont exécutés sous contrôle des interruptions et minuteries de dispositif CIA nº 2 6526. La microplaquette 6526 crée les demandes NMI (interruption non invalidable) pour le traitement RS-232. Cette caractéristique permet le traitement RS-232 non prioritaire pendant les programmes en BASIC et en langage machine. Des attentes sont intégrées dans les programmes de routine de KERNAL, de cassette et de bus série pour éviter de perturber le stockage ou la transmission des données par des interruptions non invalidables créées par les programmes RS-232. Pendant les activités de bus série ou de cassette, ON NE PEUT PAS recevoir de données des dispositifs RS-232. Ces attentes n'étant que locales (en supposant que la programmation a été soigneusement faite), il ne doit se produire aucune perturbation.

L'interface RS-232 de Commodore 64 contient deux tampons qui contribuent à éviter la perte de données pendant la transmission ou la réception des informations de RS-232.

Le KERNAL RS-232 de Commodore 64 comprend deux tampons premier entré/premier sorti, en haut de mémoire, ayant chacun 256 octets de long. Si l'on ouvre (OPEN) un canal RS-232, on affecte automatiquement 512 octets de mémoire à ces tampons. S'il n'y a pas assez d'espace libre après la fin du programme BASIC, aucun message d'erreur n'est imprimé; la fin du programme est alors détruite; **IL FAUT DONC FAIRE TRÈS ATTENTION!**

Ces tampons sont automatiquement supprimés si l'on utilise la commande CLOSE.

OUVERTURE D'UN CANAL RS-232

On ne doit ouvrir qu'un seul canal RS-232 à la fois; une deuxième instruction OPEN amène la remise à l'état initial des pointeurs de tampon. Les caractères présents dans le tampon d'émission ou le tampon de réception sont alors perdus.

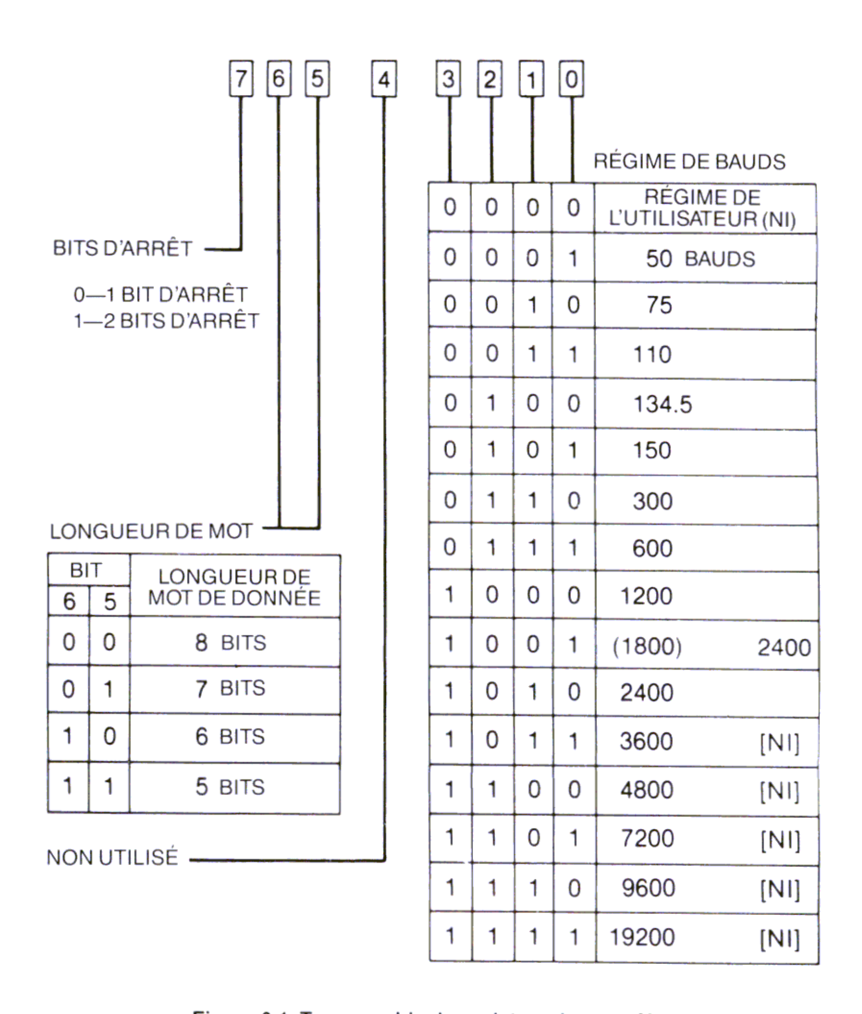
On peut envoyer jusqu'à 4 caractères dans la zone de nom de fichier. Les deux premiers correspondent aux caractères de registre de contrôle et de commande; les deux autres sont réservés aux options futures du système. Cette caractéristique permet de choisir le régime de bauds, la parité et d'autres options.

Il ne se fait aucune vérification d'erreur sur le mot de contrôle pour détecter un régime de bauds qui n'est pas mis en oeuvre. Un mot de contrôle interdit amène la sortie du système à fonctionner à très bas régime (au-dessous de 50 bauds).

SYNTAXE BASIC

OPEN lfn,2,0," <registre de contrôle> <registre de commande> <baud bas en option> <baud haut en option> "

lfn—Le numéro de fichier logique (lfn) peut être tout nombre de 0 à 255. Toutefois, si l'on choisit un numéro de fichier logique supérieur à 127, il faut savoir qu'une avance de ligne suit tous les retours de chariot.



BIT 6	BIT 5	LONGUEUR DE MOT DE DONNÉE
0	0	8 BITS
0	1	7 BITS
1	0	6 BITS
1	1	5 BITS

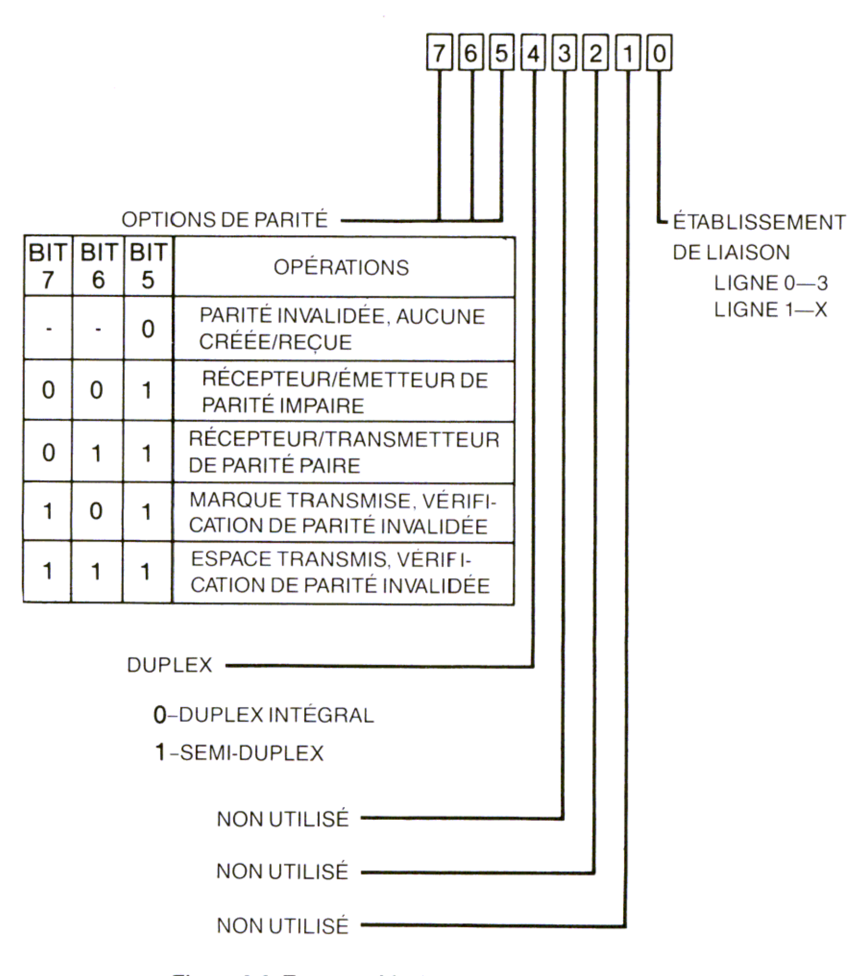
3	2	1	0	RÉGIME DE BAUDS	
0	0	0	0	RÉGIME DE L'UTILISATEUR (NI)	
0	0	0	1	50 BAUDS	
0	0	1	0	75	
0	0	1	1	110	
0	1	0	0	134.5	
0	1	0	1	150	
0	1	1	0	300	
0	1	1	1	600	
1	0	0	0	1200	
1	0	0	1	(1800)	2400
1	0	1	0	2400	
1	0	1	1	3600	[NI]
1	1	0	0	4800	[NI]
1	1	0	1	7200	[NI]
1	1	1	0	9600	[NI]
1	1	1	1	19200	[NI]

Figure 6-1. Topographie de registres de contrôle

<registre de contrôle>—Correspond à un caractère d'un octet (voir figure 6-1, topographie de registres de contrôle) requis pour spécifier les régimes de bauds. Si les 4 bits inférieurs du régime de bauds sont égaux à zéro (0), les caractères <baud bas en option> <baud haut en option> donnent un régime basé sur la formule suivante:

<baud bas en option> = <fréquence du système/régime/2—100—<baud haut en option> *256

<baud haut en option> =INT((fréquence du système/régime/2—100)/256



BIT 7	BIT 6	BIT 5	OPÉRATIONS
-	-	0	PARITÉ INVALIDÉE, AUCUNE CRÉÉE/REÇUE
0	0	1	RÉCEPTEUR/ÉMETTEUR DE PARITÉ IMPAIRE
0	1	1	RÉCEPTEUR/TRANSMETTEUR DE PARITÉ PAIRE
1	0	1	MARQUE TRANSMISE, VÉRIFICATION DE PARITÉ INVALIDÉE
1	1	1	ESPACE TRANSMIS, VÉRIFICATION DE PARITÉ INVALIDÉE

Figure 6-2. Topographie de registres de commande

Les formules ci-dessus s'appuient sur les valeurs suivantes:

fréquence du système = 1.02273E6 NTSC (normes de télévision nord-américaines)
= 0.98525E6 PAL (normes de télévision du Royaume-Uni et de la plupart des pays européens)

<registre de commande>—Correspond à un caractère d'un octet (voir figure 6-2, topographie de registres de commande) qui définit d'autres paramètres de terminal. Ce caractère N'EST PAS requis.

ENTRÉE DE KERNAL

OPEN ($FFC0) (Voir les spécifications de KERNAL pour plus de détails sur les conditions et instructions d'introduction).

> **REMARQUE IMPORTANTE:** Dans un programme BASIC, la commande OPEN RS-232 doit être exécutée avant de créer des variables ou des tableaux, car il se produit un effacement (CLR) automatique quand on ouvre un canal RS-232 (dû à l'affectation de 512 octets en haut de la mémoire). Ne pas oublier non plus que le programme est éliminé si 512 octets d'espace ne sont pas disponibles au moment de l'instruction OPEN.

EXTRACTION DE DONNÉES D'UN CANAL RS-232

Quand on extrait des données d'un canal RS-232, le tampon récepteur du Commodore 64 garde jusqu'à 255 caractères avant d'arriver au dépassement de capacité. Cette situation est indiquée par le mot d'état de RS-232 (ST en BASIC ou RSSTAT en langage machine). S'il se produit un dépassement de capacité, tous les caractères reçus quand le tampon est plein sont alors perdus. Il est donc bon de maintenir le tampon dégagé au maximum.

Si l'on veut recevoir des données RS-232 à grande vitesse (Le BASIC ne peut aller qu'à un certain régime, surtout avec la récupération des positions inutilisées. Le tampon récepteur arrive alors en dépassement de capacité), on doit utiliser des programmes de routine en langage machine pour manipuler ce type de rafales de données.

SYNTAXE BASIC:

Conseillée: GET#lfn, <variable en chaîne>
Déconseillée: INPUT#lfn, <liste de variables>

INTRODUCTIONS DE KERNAL:

CHKIN ($FFC6)—Voir la topographie de mémoire pour plus de détails sur les conditions d'introduction et de sortie.

GETIN ($FFE4)—Voir la topographie de mémoire pour plus de détails sur les conditions d'entrée et de sortie.

CHRIN ($FFCF)—Voir la topographie de mémoire pour plus de détails sur les conditions d'entrée et de sortie.

> **REMARQUES:**
>
> Si la longueur de mot est inférieure à 8 bits, tous les bits inutilisés reçoivent une valeur de 0.
>
> Si une instruction GET# ne trouve aucune donnée dans le tampon, le caractère ""(vide) est retourné.
>
> Si on utilise une instruction INPUT#, le système se met alors en attente jusqu'à la réception d'un caractère non vide et d'un retour de chariot. De ce fait, si la ligne prêt à l'envoi (CTS) ou DataSsette prêt (DSR) disparaît pendant l'entrée (INPUT#) de caractère, le système se met en état de rétablissement seulement (RESTORE). Pour cette raison, INPUT# et les programmes de routine CHRIN sont déconseillés.
>
> Le programme de routine CHKIN traite l'établissement de liaison de ligne x, conforme à la norme EIA (août 1979) pour les interfaces RS-232-C. (Les lignes "demande d'envoi" (RTS) "prêt à l'envoi" (CTS) et "signal de ligne reçue" (DCD) sont en application avec l'ordinateur Commodore 64 défini comme dispositif terminal de données.)

ENVOI DES DONNÉES À UN CANAL RS-232

Pendant l'envoi des données, le tampon de sortie peut recevoir 255 caractères avant d'arriver à un arrêt de tampon plein. Le système attend dans le programme de routine CHROUT jusqu'à ce que la transmission puisse se faire ou que l'on utilise les touches RUN/STOP et RESTORE pour rétablir le système par un "démarrage à chaud".

SYNTAXE BASIC:

CMD lfn—même rôle que dans les spécifications BASIC.

PRINT #lfn, <liste de variables>

ENTRÉES KERNAL

CHKOUT ($FFC9)—Voir la topographie de mémoire pour plus de détails sur les conditions d'entrée et de sortie.

CHROUT ($FFD2)—Voir la topographie de mémoire pour plus de détails sur les conditions d'entrée.

REMARQUES IMPORTANTES: Aucun retard de retour de chariot n'est intégré au canal de sortie. De ce fait, une imprimante RS-232 normale ne peut pas imprimer correctement, sauf si une certaine forme d'attente (demande d'attente au Commodore 64) ou un tampon interne est mis en action par l'imprimante. On peut facilement intégrer l'attente dans un programme. Si un message de liaison CTS (ligne x) est mis en vigueur, le tampon du Commodore 64 se remplit et retient la sortie ultérieure jusqu'à ce qu'une transmission soit permise par le dispositif RS-232. Le message de liaison de ligne x est un programme de routine d'établissement de liaison faisant usage de lignes multiples pour la réception et l'émission des données.

Le programme de routine CHKOUT traite le message de liaison de ligne x qui est conforme à la norme EIA (août 1979) pour les interfaces RS-232-C. Les lignes RTS, CTS et DCD sont mises en vigueur, avec le Commodore 64 défini comme dispositif terminal de données.

FERMETURE D'UN CANAL DE DONNÉES RS-232

La fermeture d'un fichier RS-232 élimine toutes les données dans les tampons au moment de l'exécution (qu'elles aient été ou non transmises ou imprimées). Elle arrête l'émission et la réception RS-232, fixe les lignes RTS et de données transmises (S_{sortie}) à l'état haut et coupe les deux tampons RS-232.

SYNTAXE BASIC:

CLOSE lfn

ENTRÉE KERNAL:

CLOSE ($FFC3)—Voir la topographie de mémoire pour plus de détails sur les conditions d'entrée et de sortie.

REMARQUE: Veiller à s'assurer que toutes les données sont transmises avant de fermer un canal. En BASIC, on peut le vérifier avec les lignes suivantes:

```
100 SS=ST: IF(SS=0 OR SS=8) THEN 100
110 CLOSE lfn
```

Table 6-1. Lignes d'accès d'utilsateur

(DISPOSITIF 6526 N° 2, positions $DD00—$DD0F)						
IDENT. BROCHE	IDENT. 6526	DESCRIPTION	EIA	ABR	ENTRÉE/ SORTIE	MODES
C	PBO	DONNÉES REÇUES	(BB)	$S_{ent.}$	ENTRÉE	1 2
D	PB1	DEMANDE D'ENVOI	(CA)	RTS	SORTIE	1*2
E	PB2	TERMINAL DE DONNÉES PRÊT	(CD)	DTR	SORTIE	1*2
F	PB3	INDICATEUR D'ANNEAU	(CE)	RI	ENTRÉE	3
H	PB4	SIGNAL DE LIGNE REÇUE	(CF)	DCD	ENTRÉE	2
J	PB5	NON ATTRIBUÉE	( )	XXX	ENTRÉE	3
K	PB6	PRÊT À L'ENVOI	(CB)	CTS	ENTRÉE	2
L	PB7	JEU DE DONNÉES PRÊT	(CC)	DSR	ENTRÉE	2
B	FLAG2	DONNÉES REÇUES	(BB)	$S_{ent.}$	ENTRÉE	1 2
M	PA2	DONNÉES TRANSMISES	(BA)	$S_{sor.}$	SORTIE	1 2
A	MASSE	MASSE DE PROTECTION	(AA)	MASSE		1 2
N	MASSE	MASSE DE SIGNAL	(AB)	MASSE		1 2 3

MODES:
1) INTERFACE DE LIGNE 3 ($S_{entrée}$, S_{sortie}, MASSE)
2) INTERFACE DE LIGNE X
3) DISPONIBLE POUR L'UTILISATEUR SEULEMENT (Non utilisé/non mis en vigueur en code.)
*Ces lignes sont maintenues à l'état haut pendant le mode de ligne 3.

Figure 6-3. Registre d'état de RS-232

```
[7]  [6]  [5]  [4]  [3]  [2]  [1]  [0]  (Langage machine — RSSATT)
 :    :    :    :    :    :    :    :__ BIT D'ERREUR DE PARITÉ
 :    :    :    :    :    :    :_______ BIT D'ERREUR D'ENCADREMENT
 :    :    :    :    :    :____________ BIT DE DÉPASSEMENT DE TAMPON
                                        RÉCEPTEUR
 :    :    :    :    :_________________ TAMPON RÉCEPTEUR—VIDE (S'UTILISE
                                        POUR VÉRIFIER APRÈS GET#)
 :    :    :    :______________________ BIT DE SIGNAL CTS MANQUANT
 :    :    :___________________________ BIT NON UTILISÉ
 :    :________________________________ BIT DE SIGNAL DSR MANQUANT
 :_____________________________________ BIT INTERRUPTION DÉTECTÉE
```

REMARQUES:

Si le bit est égal à 0, aucune erreur n'a été détectée.

On peut lire le registre d'état de RS-232 à partir du BASIC avec la variable ST.

Si ST est lu par le BASIC ou en utilisant le programme de routine KERNAL READST, le mot d'état de RS-232 est effacé à la sortie. En cas d'utilisation multiple du mot d'état, attribuer ST à une autre variable. Par exemple:

SR=ST: REM AFFECTE ST À SR

L'état de RS-232 n'est lu (et effacé) que si le canal RS-232 a été la dernière entrée/sortie extérieure utilisée.

EXEMPLES DE PROGRAMMES BASIC

```
10 REM CE PROGRAMME ASSURE LA COMMUNICATION DES DONNEES AVEC UN TERMINAL
11 REM SILENCIEUX 700 MODIFIE POUR ASCII DE PET
20 REM ETABLISSEMENT DU 700 SILENCIEUX: 300 BAUDS, ASCII 7 BITS, PARITE DE MARQUE,
21 REM DUPLEX INTEGRAL
30 REM DISPOSITION A L'ORDINATEUR AVEC INTERFACE DE LIGNE 3
100 OPEN 2,2,3,CHR$(6+32)+CHR$(32+128):REM OUVRE LE CANAL
110 GET#2,A$:REM MET LE CANAL RECEPTEUR EN FONCTION (ENVOIE UN VIDE)
200 REM BOUCLE PRINCIPALE
210 GET B$:REM LIT LE CLAVIER DE L'ORDINATEUR
220 IF B$<>"" THEN PRINT#2,B$;:REM SI UNE TOUCHE A ETE PRESSEE, ENVOYER AU TERMINAL
230 GET#2,C$:REM LIT UNE TOUCHE DU TERMINAL
240 PRINT B$;C$;:REM IMPRIME TOUTES LES ENTREES SUR L'ECRAN DE L'ORDINATEUR
250 SR=ST: IF SR=0 OR SR=8 THEN 200:REM VERIFIE L'ETAT, S'IL EST BON, CONTINUE
300 REM INDICATION D'ERREUR
310 PRINT"ERREUR: ";
320 IF SR AND 1 THEN PRINT "PARITE"
330 IF SR AND 2 THEN PRINT "TRAME"
340 IF SR AND 4 THEN PRINT "TAMPON DE RECEPTEUR PLEIN"
350 IF SR AND 128 THEN PRINT "INTERRUPTION"
360 IF (PEEK(673) AND 1) THEN 360:REM ATTEND JUSQU'A LA TRANSMISSION DE TOUS LES CARACTERES
370 CLOSE 2: END
```

```
10 REM CE PROGRAMME COMMUNIQUE LES DONNEES ASCII VRAIES
100 OPEN 5,2,3,CHR$(6)
110 DIM F%(255),T%(255)
200 FOR J=32 TO 64:T%(J)=J:NEXT
210 T%(13)=13:T%(20)=8:RV=18:CT=0
220 FOR J=65 TO 90:K=J+32:T%(J)=K:NEXT
230 FOR J=91 TO 95:T%(J)=J:NEXT
240 FOR J=193 TO 218:K=J-128:T%(J)=K:NEXT
250 T%(146)=16:T%(133)=16
260 FOR J=0 TO 255
270 K=T%(J)
280 IF K<>0THEN F%(K)=J:F%(K+128)=J
290 NEXT
300 PRINT " "CHR$(147)
310 GET#5,A$
320 IF A$=""OR ST<>0 THEN 360
330 PRINT " "CHR$(157);CHR$(F%(ASC(A$)));
340 IF F%(ASC(A$))=34 THEN POKE212,0
350 GOTO 310
360 PRINTCHR$(RV)" "CHR$(157);CHR$(146);:GET A$
370 IF A$<>""THENPRINT#5,CHR$(T%(ASC(A$)));
380 CT=CT+1
390 IF CT=8 THENCT=0:RV=164-RV
410 GOTO310
```

POINTEURS DE POSITION DE BASE DE TAMPON DE RÉCEPTEUR/ÉMETTEUR

$00F7—RIBUF—Pointeur à deux octets de la position de base de tampon de récepteur.

$00F9—ROBUF—Pointeur à deux octets de la position de base de tampon d'émetteur.

Les deux positions ci-dessus sont établies par le programme de routine KERNAL OPEN; chacune est pointée vers un tampon différent de 256 octets. Elles sont désaffectées par l'écriture d'un zéro dans les octets de poids fort ($00F8 et $00FA), qui se fait avec l'entrée CLOSE de KERNAL. Le programmeur en langage machine peut aussi affecter/désaffecter à ses propres fins en n'enlevant ou créant que les tampons requis. Si l'on utilise un programme en langage machine qui attribue ces tampons, il ne faut pas oublier de s'assurer que le haut des pointeurs de mémoire reste correct, en particulier si l'on compte exécuter des programmes BASIC en même temps.

POSITIONS ET UTILISATION DE MÉMOIRE DE PAGE ZÉRO POUR L'INTERFACE DE SYSTÈME RS-232

$00A7—INBIT—Stockage temporaire de bit d'entrée de récepteur.

$00A8—BITCI—Début comptage bit de récepteur.

$00A9—RINONE—Vérification de bit de départ d'indicateur de récepteur.

$00AA—RIDATA—Position d'assemblage/tampon d'octet de récepteur.

$00AB—RIPRTY—Stockage de bit de parité de récepteur.

$00B4—BITTS—Fin comptage bit d'émetteur.

$00B5—NXTBIT—Bit suivant d'émetteur à envoyer.

$00B6—RODATA—Position de démontage/tampon d'octet d'émetteur.

Toutes les positions de page zéro ci-dessus servent localement et ne sont données qu'à titre de guide pour comprendre les programmes de routine connexes. Le programmeur au niveau BASIC ou KERNAL ne peut pas les utiliser directement pour procéder à des opérations de type RS-232. On doit utiliser les programmes de routine de système RS-232.

POSITIONS ET UTILISATION DE MÉMOIRE DE PAGE DIFFÉRENTE DE ZÉRO POUR L'INTERFACE DE SYSTÈME RS-232

Stockage général RS-232:

$0293—M51CTR—Pseudo-registre de contrôle 6551 (voir figure 6-1).

$0294—M51COR—Pseudo-registre de commande 6551 (voir figure 6-2).

$0295—M51AJB—Deux octets à la suite des registres de contrôle et de commande dans la zone de nom de fichier. Ces positions contiennent le régime de bauds de début et la vérification de bit pendant l'activité de l'interface qui sert à son tour à calculer le régime de bauds.

$0297—RSSTAT—Registre d'état de RS-232 (voir figure 6-3).

$0298—BITNUM—Nombre de bits reçus à envoyer.

$0299—BAUDOF—Deux octets égaux à la durée d'une cellule binaire. (D'après l'horloge du système/régime de bauds.)

$029B—RIDBE—Index d'octet à la fin du tampon premier entré/premier sorti de récepteur.

$029C—RIDBS—Index d'octet au début du tampon premier entré/premier sorti de récepteur.

$029D—RODBS—Index d'octet au début du tampon premier entré/premier sorti d'émetteur.

$029E—RODBE—Index d'octet à la fin du tampon premier entré/premier sorti d'émetteur.

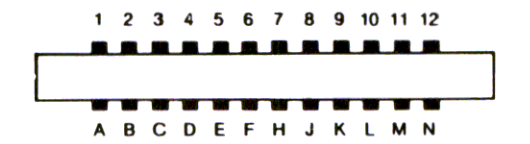
$02A1—ENABL—Contient des interruptions actives présentes dans l'ICR de la CIA n° 2. Quand le bit 4 est mis à un, le système attend l'impulsion de récepteur. Quand le bit 1 est mis à un, le système reçoit alors les données. Quand le bit 0 est mis à un, le système émet les données.

ACCÈS D'UTILISATEUR

L'accès d'utilisateur sert à raccorder le Commodore 64 à l'extérieur. À l'aide des lignes disponibles à cet accès, on peut raccorder le Commodore 64 à une imprimante, à un dispositif "Type and Talk" de Votrax, à un MODEM et même à un autre ordinateur.

L'accès du Commodore 64 est directement raccordé à l'une des microplaquettes CIA 6526. Au moyen de la programmation, la CIA se raccorde à de nombreux autres dispositifs.

BROCHAGE DE L'ACCÈS



DESCRIPTION DU BROCHAGE DE L'ACCÈS

BROCHE CÔTÉ SUPÉRIEUR	DESCRIPTION	REMARQUES
1	MASSE	
2	+5 V	(100 mA max.)
3	REMISE À L'ÉTAT INITIAL	En mettant cette broche à la masse, le Commodore 64 procède à un démarrage à froid, avec remise complète à l'état initial. Les pointeurs d'un programme BASIC sont remis à l'état initial, mais la mémoire n'est pas effacée. On obtient aussi une sortie de remise à l'état initial pour les dispositifs extérieurs.
4	CNT1	Compteur d'accès série de la CIA nº 1 (voir spécifications de la CIA).
5	SP1	Accès série de la CIA nº 1 (voir spécifications de la CIA 6526).
6	CNT2	Compteur d'accès série de la CIA nº 2 (voir spécifications de la CIA).
7	SP2	Accès série de la CIA nº 1 (voir spécifications de la CIA 6526).
8	PC2	Ligne de message de liaison de la CIA nº 2 (voir spécifications de la CIA).
9	ATN SÉRIE	Cette broche est reliée à la ligne ATN du bus série.
10	CA 9 V + phase	Raccordée directement au transformateur du Commodore 64 (50 mA max.).
11	CA 9 V − phase	
12	MASSE	
DESSOUS		
A	MASSE	Le Commodore 64 permet le contrôle de l'accès B sur la microplaquette CIA nº 1. On dispose de huit lignes pour l'entrée ou la sortie ainsi que de 2 lignes pour le message de liaison avec un dispositif extérieur. Les lignes d'entrée/sortie de l'accès B sont commandées par deux positions. L'une, qui correspond à l'accès lui-même, se trouve à 56577 ($DD01 hex). On utilise une instruction PEEK pour lire une introduction (INPUT) ou POKE pour fixer une sortie (OUTPUT). On peut établir chacune des huit lignes d'entrée/sortie pour l'entrée ou la sortie en fixant convenablement le registre de direction des données.
B	INDICATEUR 2	
C	PBO	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	MASSE	

La **position du registre de direction des données** se trouve à 56579 ($DD03 en hexadécimal). Chacune des huit lignes de l'accès possède un bit dans le registre de direction des données (DDR) à huit bits qui décide si la ligne doit être une entrée ou une sortie. Si un bit dans le registre de direction est à un, la ligne correspondante de l'accès est une sortie; si un bit du registre est à zéro, la ligne correspondante de l'accès est une entrée. Par exemple, si le bit 3 du registre de direction est à 1, la ligne 3 de l'accès est alors une sortie. Prenons un autre exemple:

Si le registre de direction DDR est fixé de la façon suivante:

BIT n°:	7	6	5	4	3	2	1	0
VALEUR:	0	0	1	1	1	0	0	0

On peut voir que les lignes 5, 4, et 3 sont des sorties car les bits correspondants sont à 1. Les autres lignes sont des entrées, car les bits correspondants sont à 0.

Pour lire (PEEK) ou écrire (POKE) dans l'accès d'utilisateur, on doit d'abord utiliser le registre de direction de données (DDR) et l'accès lui-même.

Ne pas oublier que les instructions PEEK et POKE s'accompagnent d'un nombre de 0 à 255. Les nombres donnés dans l'exemple ci-dessous doivent être convertis en notation décimale avant de pouvoir les utiliser. On doit obtenir la valeur:

$$2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$$

On peut remarquer que le n° de bit pour le registre DDR est le même que 2 élevé à une puissance pour mettre la valeur de bit en fonction.

(16 = 2 ↑ 4=2X2X2X2, 8 = 2 ↑ 3=2X2X2)

Les deux autres lignes, INDICATEUR1 et PA2, diffèrent du reste de l'accès d'utilisateur. Ces deux lignes, qui servent essentiellement à l'établissement de la liaison, sont programmées différemment de l'accès B.

On doit établir la liaison quand deux dispositifs communiquent. Un dispositif peut fonctionner à une vitesse différente d'un autre; il faut alors que chaque dispositif soit au courant des dispositions de l'autre. Si les dispositifs fonctionnent à la même vitesse, on doit quand même établir la liaison pour que l'autre sache s'il y a des données à envoyer et si elles ont été reçues. La ligne **INDICATEUR1** possède des caractéristiques spéciales qui la rendent particulièrement utile pour l'établissement de la liaison.

INDICATEUR1 est une entrée à impulsion négative que l'on peut utiliser comme entrée d'interruption de type général. Une transition négative sur la ligne d'indicateur met le bit d'interruption d'indicateur à un. Si l'interruption d'indicateur est validée, elle amène une demande d'interruption. Si le bit d'indicateur n'est pas validé, il peut être interrogé depuis le registre d'interruption, sous contrôle du programme.

PA2 correspond au bit 2 de l'accès A de la microplaquette CIA. Il est contrôlé comme les autres bits de l'accès. L'accès se trouve à la position 56576 ($DD00). Le registre de direction des données se trouve à 56578 ($DD02).

POUR PLUS DE DÉTAILS SUR LA 6526, VOIR LES SPÉCIFICATIONS DE MICRO-PLAQUETTE DANS L'ANNEXE M.

BUS SÉRIE

Le bus série est une disposition en guirlande qui permet au Commodore 64 de communiquer avec des dispositifs comme l'unité de disque VIC-1541 et l'imprimante graphique VIC-1525. Le bus série présente l'avantage de permettre le raccordement de plusieurs dispositifs à l'accès. On peut brancher jusqu'à 5 dispositifs à la fois au bus série.

Il existe trois types d'opérations dans un bus série: commande, émission et écoute. Un dispositif contrôleur surveille le fonctionnement du bus série. Un émetteur transmet les données dans le bus. Un écouteur reçoit les données du bus.

Le Commodore 64 joue le rôle de contrôleur du bus. Il sert aussi d'émetteur (par exemple, envoi des données à l'imprimante) et d'écouteur (par exemple, chargement d'un programme venant de l'unité de disque). On peut utiliser d'autres dispositifs remplissant séparément les fonctions d'écouteur (imprimante), d'émetteur ou les deux à la fois (unité de disque). Le Commodore 64 peut seul servir de contrôleur.

Tous les dispositifs raccordés au bus série reçoivent toutes les données qui y sont transmises. Pour que le Commodore 64 achemine les données à leur destination prévue, chaque dispositif possède une adresse de bus. Grâce à cette adresse, le Commodore 64 peut commander l'accès au bus. Sur le bus série, les adresses vont de 4 à 31.

Le Commodore 64 peut commander l'émission ou la réception d'un dispositif particulier. Quand le Commodore 64 "ordonne" à un dispositif d'émettre, celui-ci injecte les données dans le bus série. Si le Commodore 64 "ordonne" à un dispositif d'écouter, celui-ci se prépare à recevoir les données (du Commodore 64 ou d'un autre dispositif raccordé au bus). Un seul dispositif à la fois peut émettre sur le bus, sinon les données entrent en collision et il se produit un incident de système. Plusieurs dispositifs peuvent cependant écouter simultanément un émetteur.

ADRESSES COURANTES DE BUS SÉRIE

NUMÉRO	DISPOSITIF
4 ou 5	IMPRIMANTE GRAPHIQUE VIC-1525
8	UNITÉ DE DISQUE VIC-1541

On peut avoir d'autres adresses de dispositif. Chaque dispositif doit avoir sa propre adresse. Certains dispositifs, comme l'imprimante de Commodore 64, permettent le choix entre deux adresses, pour faciliter les choses à l'utilisateur.

L'adresse secondaire permet au Commodore 64 de transmettre des informations de préparation à un dispositif. Par exemple, pour ouvrir (OPEN) une liaison sur le bus, en direction de l'imprimante et pour que celle-ci imprime en majuscules/minuscules, utiliser la ligne suivante:

OPEN 1,4,7

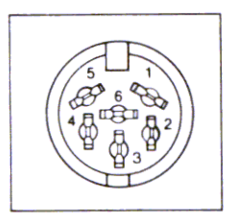
dans laquelle:

1 représente le numéro de fichier logique (numéro vers lequel on imprime — PRINT#),
4 représente l'adresse de l'imprimante et
7 représente l'adresse secondaire qui indique à l'imprimante de passer en mode de majuscules/minuscules.

On utilise 6 lignes dans le bus série: 3 pour l'entrée et 3 pour la sortie. Les 3 lignes d'entrée amènent les signaux de données, de commande et de synchronisation au Commodore 64. Les 3 lignes de sortie envoient les signaux de données, de commande et de synchronisation du Commodore 64 aux dispositifs extérieurs raccordés au bus série.

BROCHAGE DU BUS SÉRIE

BROCHE	DESCRIPTION
1	ENTRÉE SRQ SÉRIE
2	MASSE
3	ENTRÉE/SORTIE ATN SÉRIE
4	ENTRÉE/SORTIE HORLOGE SÉRIE
5	ENTRÉE/SORTIE DONNÉES SÉRIE
6	NON CONNECTÉ



ENTRÉE SRQ SÉRIE (ENTRÉE DE DEMANDE DE SERVICE SÉRIE):

Tout dispositif sur le bus série peut amener l'état BAS pour demander l'attention du Commodore 64. Ce dernier prend alors ce dispositif en charge (voir figure 6-4).

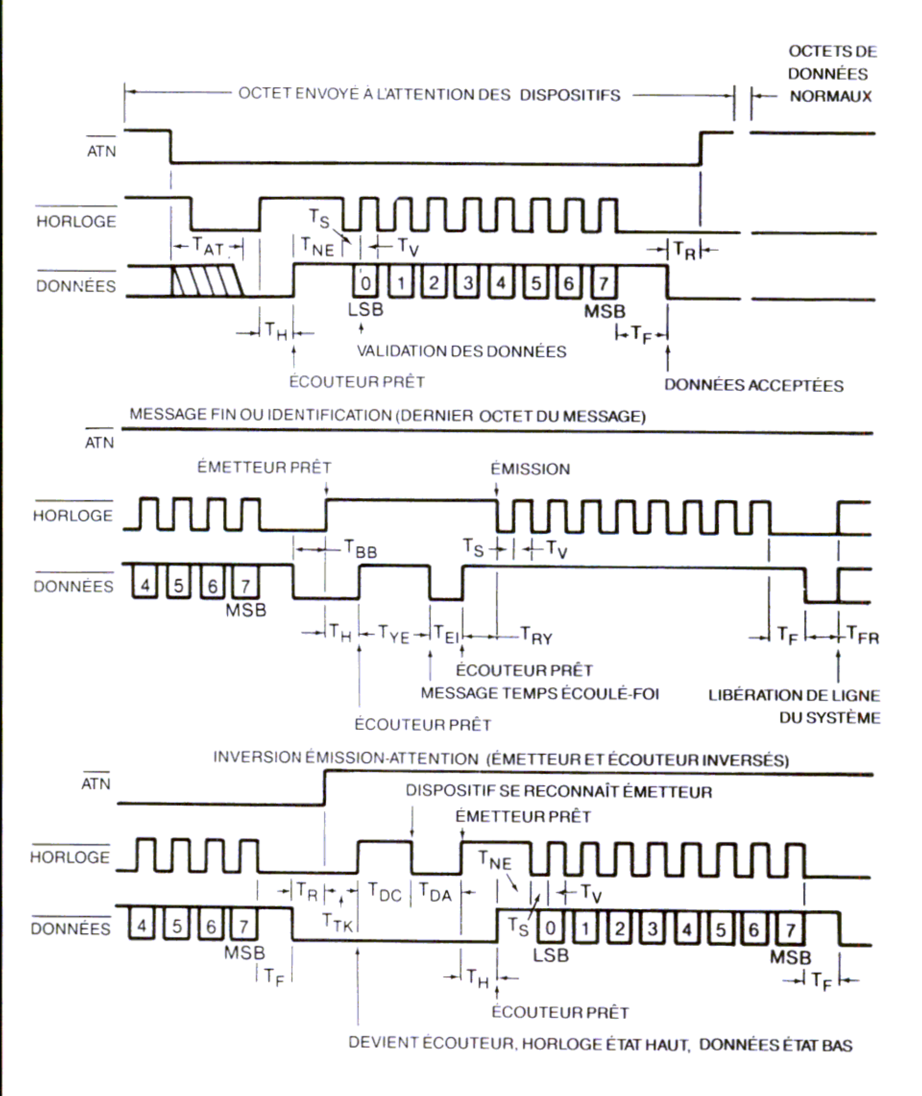
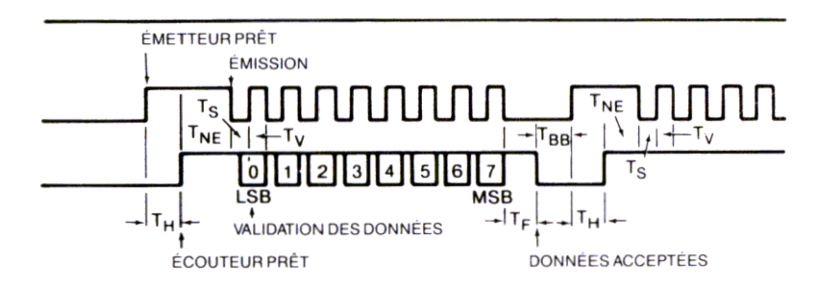


Figure 6-4. Synchronisation du bus série

ENTRÉE/SORTIE ATN SÉRIE (ENTRÉE/SORTIE ATTENTION SÉRIE):

Le Commodore 64 utilise ce signal pour déclencher une séquence de commandes pour un dispositif sur le bus série. Lorsque le Commodore 64 amène l'état BAS, tous les autres dispositifs sur le bus se mettent en position d'écoute, attendant l'émission d'une adresse par le Commodore 64. Le dispositif se trouvant à l'adresse en question doit répondre dans un délai fixé; sinon, le Commodore 64 suppose qu'il ne se trouve pas sur le bus et envoie un message d'erreur dans le MOT D'ÉTAT (voir figure 6-4).



SYNCHRONISATION DU BUS SÉRIE

Description	Symbole	Min.	Typ.	Max.
RÉPONSE ATN (REQUISE)[1]	T_{AT}	—	—	1000 µs
DÉLAI ÉCOUTEUR	T_H	0	—	∞
RÉPONSE NON-FOI À ÉCOUTEUR PRÊT[2]	T_{NE}	—	40 µs	200 µs
BIT DE PRÉPARATION D'ÉMETTEUR[4]	T_S	20 µs	70 µs	—
VALIDATION DE DONNÉES	T_V	20 µs	20 µs	—
LIAISON DE TRAME[3]	T_F	0	20	1000 µs
DÉLAI TRAME/LIBÉRATION DE L'ATTENTION	T_R	20 µs	—	—
INTERVALLE ENTRE OCTETS	T_{BB}	100 µs	—	—
TEMPS DE RÉPONSE FOI	T_{YE}	200 µs	250 µs	—
DÉLAI DE LA RÉPONSE FOI[5]	T_{EI}	60 µs	—	—
DÉLAI MAX. DE RÉPONSE DE L'ÉMETTEUR	T_{RY}	0	30 µs	60 µs
ACCUSÉ DE RÉCEPTION D'OCTET[4]	T_{PR}	20 µs	30 µs	—
LIBÉRATION ÉMISSION-ATTENTION	T_{TK}	20 µs	30 µs	100 µs
ACCUSÉ DE RÉCEPTION ÉMISSION-ATTENTION	T_{DC}	0	—	—
DÉLAI D'ACCUSÉ DE RÉCEPTION ÉMISSION-ATTENTION	T_{DA}	80 µs	—	—
ACCUSÉ DE RÉCEPTION FOI	T_{FR}	60 µs	—	—

Remarques:

1. Si le temps maximal est dépassé, le dispositif n'affiche pas d'erreur.
2. Si le temps maximal est dépassé, il faut une réponse FOI
3. Si le temps maximal est dépassé, il y a erreur de trame.
4. Les T_V et les T_{PR} minimaux doivent être de 60 µs pour qu'un dispositif extérieur soit émetteur.
5. Le T_{EI} minimal doit être de 80 µs pour que le dispositif extérieur soit écouteur.

ENTRÉE/SORTIE HORLOGE SÉRIE:

Ce signal sert à la synchronisation des données envoyées dans le bus série (voir figure 6-4).

ENTRÉE/SORTIE DE DONNÉES SÉRIE:

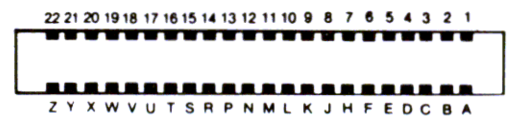
Dans le bus série, les données sont transmises à raison d'un bit à la fois sur cette ligne. (Voir figure 6-4).

ACCÈS D'EXTENSION

À l'arrière du Commodore 64, le raccord d'extension de type femelle plat possède 44 broches. Si l'on fait face au Commodore 64, le raccord d'extension se trouve à l'extrême droite, à l'arrière de l'ordinateur. Il s'utilise avec un raccord mâle plat de 44 broches (22 de chaque côté).

Cet accès sert aux extensions du système Commodore 64 qui doivent être reliées au bus d'adresse ou au bus de données de l'ordinateur. Il faut faire attention quand on utilise le bus d'extension, car un défaut de fonctionnement de l'équipement peut endommager le Commodore 64.

Le bus d'extension est disposé de la façon suivante:



On dispose les signaux suivants sur le raccord:

NOM	BROCHE	DESCRIPTION
MASSE	1	Masse du système
CC+5 V	2	(L'ensemble des dispositifs d'accès d'utilisateur et sur
CC+5 V	3	cartouche ne peuvent pas consommer plus de 450 mA.)
$\overline{\text{IRQ}}$	4	Ligne de demande d'interruption à la 6502 (état bas actif)
R/$\overline{\text{W}}$	5	Lecture/écriture
HORLOGE DE POINT	6	Horloge de point vidéo, 8.18 MHz
$\overline{\text{I/O1}}$	7	Bloc d'entrée/sortie 1 @ $DE00—$DEFF (état bas actif), entrée/sortie non tamponnée
$\overline{\text{JEU}}$	8	Entrée ttl ls à état bas actif
$\overline{\text{EXROM}}$	9	Entrée ttl ls à état bas actif
$\overline{\text{I/02}}$	10	Bloc d'entrée/sortie 2 @ $DF00—$DFFF (état bas actif) sortie ttl ls tamponnée
$\overline{\text{ROML}}$	11	Bloc RAM/ROM décodé de 8 K @ $8000 (état bas actif), sortie ttl ls tamponnée

NOM	BROCHE	DESCRIPTION	
BA	12	Signal disponible de bus de la microplaquette VIC-II, charge max. ls 1, non tamponnée	
$\overline{\text{DMA}}$	13	Ligne de demande d'accès de mémoire direct (entrée basse active) entrée ttl ls	
D7	14	Bit 7 de bus de données	
D6	15	Bit 6 de bus de données	
D5	16	Bit 5 de bus de données	
D4	17	Bit 4 de bus de données	Charge max. ttl ls
D3	18	Bit 3 de bus de données	1, non tamponnée
D2	19	Bit 2 de bus de données	
D1	20	Bit 1 de bus de données	
D0	21	Bit 0 de bus de données	
MASSE	22	Masse du système	
MASSE	A		
$\overline{\text{ROMH}}$	B	Bloc RAM/ROM décodé de 8 K @ $E000, tamponné	
$\overline{\text{RESET}}$	C	Broche de remise à l'état initial de 6502 (état bas actif), sortie ttl tamponnée/entrée non tamponnée	
$\overline{\text{NMI}}$	D	Interruption non invalidable de 6502 (état bas actif), sortie ttl tamponnée, entrée non tamponnée	
ϕ2	E	Horloge de système, phase 2	
A15	F	Bit 15 de bus d'adresse	
A14	H	Bit 14 de bus d'adresse	
A13	J	Bit 13 de bus d'adresse	
A12	K	Bit 12 de bus d'adresse	
A11	L	Bit 11 de bus d'adresse	
A10	M	Bit 10 de bus d'adresse	Charge max. ttl ls
A9	N	Bit 9 de bus d'adresse	1, non tamponnée
A8	P	Bit 8 de bus d'adresse	
A7	R	Bit 7 de bus d'adresse	
A6	S	Bit 6 de bus d'adresse	
A5	T	Bit 5 de bus d'adresse	
A4	U	Bit 4 de bus d'adresse	
A3	V	Bit 3 de bus d'adresse	
A2	W	Bit 2 de bus d'adresse	
A1	X	Bit 1 de bus d'adresse	
A0	Y	Bit 0 de bus d'adresse	
MASSE	Z	Masse du système	

Le trait sur les noms correspond à l'état bas actif.

Nous expliquons ci-dessous quelques points importants de l'accès d'extension:

Les broches 1, 22, A et Z sont raccordées à la masse du système.

La broche 6 correspond à l'horloge de point vidéo de 8.18 MHz. Toute la synchronisation du système vient de cette horloge.

La broche 12 correspond au signal de bus disponible (BA) de la microplaquette VIC-II. Cette ligne passe 3 cycles à l'état bas avant que la VIC-II prenne en charge les bus du système; elle reste à l'état bas jusqu'à ce que la VIC-II ait fini d'extraire les informations de l'affichage.

La broche 13 correspond à la ligne d'accès direct à la mémoire (DMA). Quand cette ligne est à l'état bas, le bus d'adresse, le bus de données et la ligne de lecture/écriture de la microplaquette 6510 passent en mode à haute impédance. Une unité de traitement extérieure peut alors contrôler les bus du système. Ne mettre cette ligne à l'état bas que si l'horloge $\phi 2$ est aussi à l'état bas. En outre, la microplaquette VIC-II continue à exécuter la ligne d'accès direct à la mémoire (DMA), le dispositif extérieur doit être synchronisé avec la VIC-II. (Voir diagramme de synchronisation de la VIC-II.) Cette ligne est décalée sur le Commodore 64.

CARTOUCHE DE MICROPROCESSEUR Z-80

La lecture de ce manuel et l'utilisation de l'ordinateur montrent la flexibilité du Commodore 64. Pour mieux satisfaire encore les besoins de l'utilisateur, on peut ajouter des équipements périphériques, comme le magnétocassette Datassette™, des unités de disque, des imprimantes et des modems. On peut raccorder tous ces appareils au Commodore 64 par les différents accès et prises de l'arrière de l'ordinateur. Les périphériques Commodore sont dits "intelligents", car ils n'occupent pas de la place précieuse de mémoire vive (RAM) quand ils sont en service. Les 64 K de mémoire du Commodore 64 restent ainsi libres.

D'autre part, la majeure partie des programmes écrits aujourd'hui avec le Commodore 64 possèdent la compatibilité ascendante avec les ordinateurs futurs que Commodore doit mettre en service. On doit cette caractéristique aux qualités du système d'exploitation de l'ordinateur.

Le système d'exploitation Commodore ne permet cependant pas de rendre les programmes compatibles avec un ordinateur fabriqué par une autre compagnie.
Dans la majeure partie des cas, on ne pensera même pas aux ordinateurs d'une autre compagnie, car le Commodore 64 est particulièrement facile à utiliser. Toutefois, pour l'utilisateur occasionnel qui désire profiter de logiciels qui ne sont pas offerts en format Commodore 64, nous avons mis au point une cartouche CP/M® de Commodore.

Le CP/M® n'est pas un système d'exploitation propre à un ordinateur. Il utilise une partie de l'espace de mémoire normalement disponible pour la programmation pour exécuter son propre système d'exploitation. Cette disposition présente des avantages et des inconvénients. Parmi les inconvénients, les programmes que l'on écrit doivent être plus courts que ceux qu'on prépare avec le système d'exploitation intégré du Commodore 64. En outre, on ne peut pas mettre à profit les puissantes possibilités d'édition d'écran du Commodore 64. Parmi les avantages, on peut utiliser dès maintenant une vaste bibliothèque de logiciels qui a été spécifiquement établie pour le CP/M® et le microprocesseur Z-80; les programmes que l'on écrit avec le système d'exploitation CP/M® peuvent être transposés et exécutés sur tout autre ordinateur doté du CP/M® et d'une carte Z-80.

Avec la plupart des ordinateurs dotés d'un microprocesseur Z-80, on doit aller à l'intérieur de l'appareil pour installer une carte Z-80. Il faut alors faire très attention de ne pas toucher aux circuits délicats du reste de l'ordinateur. La cartouche CP/M® Commodore élimine ce problème, car notre cartouche Z-80 s'installe rapidement et facilement dans l'arrière du Commodore 64, sans aucun câblage encombrant risquant de causer des problèmes par la suite.

UTILISATION DU CP/M® DE COMMODORE

La cartouche Z-80 de Commodore permet d'exécuter avec le Commodore 64 les programmes prévus pour un microprocesseur Z-80. La cartouche est fournie avec un minidisque contenant le système d'exploitation CP/M® de Commodore.

EXÉCUTION DU CP/M® DE COMMODORE

Pour exécuter le CP/M®:

1) CHARGER (LOAD) le programme CP/M® à partir de l'unité de disque.
2) Taper RUN (exécution).
3) Appuyer sur la touche RETURN

À ce stade, l'unité centrale 6510 intégrée peut accéder aux 64 K-octets de mémoire vive RAM du Commodore 64 OU l'unité centrale Z-80 peut disposer de 48 K-octets de mémoire vive RAM. Grâce au mécanisme perfectionné de synchronisation du Commodore 64, on peut passer de l'un à l'autre de ces deux microprocesseurs, mais on ne peut pas les utiliser simultanément dans un même programme.

Nous indiquons ci-dessous la traduction des adresses de mémoire exécutée par la cartouche Z-80. On peut remarquer que, en ajoutant 4096 octets ($1000 en hexadécimal) aux positions de mémoire utilisées dans le CP/M®, on obtient les adresses de mémoire du système d'exploitation normal du Commodore 64. La table suivante donne les correspondances entre les adresses de mémoire Z-80 et 6510:

ADRESSES Z-80		ADRESSES 6510	
DÉCIMAL	HEXADÉCIMAL	DÉCIMAL	HEXADÉCIMAL
0000—4095	0000—0FFF	4096—8191	1000—1FFF
4096—8191	1000—1FFF	8192—12287	2000—2FFF
8192—12287	2000—2FFF	12288—16383	3000—3FFF
12288—16383	3000—3FFF	16384—20479	4000—4FFF
16384—20479	4000—4FFF	20480—24575	5000—5FFF
20480—24575	5000—5FFF	24576—28671	6000—6FFF
24576—28671	6000—6FFF	28672—32767	7000—7FFF
28672—32767	7000—7FFF	32768—36863	8000—8FFF
32768—36863	8000—8FFF	36864—40959	9000—9FFF
36864—40959	9000—9FFF	40960—45055	A000—AFFF
40960—45055	A000—AFFF	45056—49151	B000—BFFF
45056—49151	B000—BFFF	49152—53247	C000—CFFF
49152—53247	C000—CFFF	53248—57343	D000—DFFF
53248—57343	D000—DFFF	57344—61439	E000—EFFF
57344—61439	E000—EFFF	61440—65535	F000—FFFF
61440—65535	F000—FFFF	0000—4095	0000—OFFF

Pour mettre le Z-80 en fonction et couper la 6510, taper le programme suivant:

```
10 REM CE PROGRAMME S'UTILISE AVEC LA CARTE Z80
20 REM IL STOCKE D'ABORD LES DONNEES A $1000 (Z80 A $0000)
30 REM IL COUPE ENSUITE LES INSTRUCTIONS IRQ DE LA 6510 ET VALIDE
40 REM LA CARTE Z80. LA CARTE Z80 DOIT ETRE COUPEE
50 REM POUR REVALIDER LE SYSTEME 6510
100 STOCKAGE DES DONNEES Z80
110 READ B: REM EXTRAIT LA LONGUEUR DU CODE Z80 A DEPLACER
120 FOR I=4096 TO 4096+B-1: REM DEPLACE LE CODE
130 READ A:POKE I,A
140 NEXT I
200 REM EXECUTE LE CODE Z80
210 POKE 56333,127 : REM COUPE LES INSTRUCTIONS IRQ DE 6510
220 POKE 56832,00 : REM MET LA CARTE Z80 EN FONCTION
230 POKE 56333,129 : REM MET LES INSTRUCTIONS IRQ DE 6510 EN FONCTION
QUAND Z80 EST EXECUTE
240 END
1000 REM SECTION DE DONNEES DE CODE DE LANGAGE MACHINE Z80
1010 DATA 18 : REM LONGUEUR DES DONNEES A PASSER
1100 REM CODE DE MISE EN FONCTION DE Z80
1110 DATA 00,00,00 : REM NOTRE CARTE Z80 REQUIERT UN TEMPS DE MISE EN
FONCTION A $0000
1200 REM DONNEES DE TACHE Z80 ICI
1210 DATA 33,02,245 : REM LD HL,NN (POSITION SUR L'ECRAN)
1220 DATA 52 : REM PROGRESSION HL (AUGMENTE CETTE POSITION)
1300 REM DONNEES D'ARRET AUTOMATIQUE Z80 ICI
1310 DATA 62,01 : REM LD A,N
1320 DATA 50,00,206 : REM LD (NN),A :POSITION D'ENTREE/SORTIE
1330 DATA 00,00,00 : REM NOP: NOP: NOP
1340 DATA 195,00,00 : REM SAUT $0000
```

Pour plus de détails sur le CP/M® de Commodore et le microprocesseur Z-80, se procurer la cartouche et le guide de référence Z-80 au distributeur Commodore local.

ANNEXES

ANNEXE A

ABRÉVIATIONS DES MOTS CLÉS DE BASIC

Pour gagner du temps dans la frappe des programmes et des commandes, le BASIC de Commodore 64 permet l'abréviation de la plupart des mots clés. Par exemple, PRINT s'abrège sous la forme d'un point d'interrogation. On abrège les autres mots en tapant la première ou les deux premières lettres du mot, en appuyant sur SHIFT et en ajoutant la lettre suivante. Si l'on utilise des abréviations dans une ligne de programme, le mot clé est listé sous sa forme intégrale.

Commande	Abréviation	Aspect sur l'écran	Commande	Abréviation	Aspect sur l'écran
ABS	A SHIFT B	A	END	E SHIFT N	E
AND	A SHIFT N	A	EXP	E SHIFT X	E ♣
ASC	A SHIFT S	A ♥	FN	Aucune	FN
ATN	A SHIFT T	A	FOR	F SHIFT O	F
CHR$	C SHIFT H	C	FRE	F SHIFT R	F
CLOSE	CL SHIFT O	CL	GET	G SHIFT E	G
CLR	C SHIFT L	C	GET#	Aucune	GET#
CMD	C SHIFT M	C	GOSUB	GO SHIFT S	GO ♥
CONT	C SHIFT O	C	GOTO	G SHIFT O	GO
COS	Aucune	COS	IF	Aucune	IF
DATA	D SHIFT A	D ♠	INPUT	Aucune	INPUT
DEF	D SHIFT E	D	INPUT#	I SHIFT N	I
DIM	D SHIFT I	D	INT	Aucune	INT

Commande	Abréviation	Aspect sur l'écran
LEFT$	LE SHIFT F	LE
LEN	Aucune	LEN
LET	L SHIFT E	L
LIST	L SHIFT I	L
LOAD	L SHIFT O	L
LOG	Aucune	LOG
MID$	M SHIFT I	M
NEW	Aucune	NEW
NEXT	N SHIFT E	N
NOT	N SHIFT O	N
ON	Aucune	ON
OPEN	O SHIFT P	O
OR	Aucune	OR$
PEEK	P SHIFT E	P
POKE	P SHIFT O	P
POS	Aucune	POS
PRINT	?	?
PRINT#	P SHIFT R	P
READ	R SHIFT E	R
REM	Aucune	REM
RESTORE	RE SHIFT S	RE ♥
RETURN	RE SHIFT T	RE

Commande	Abréviation	Aspect sur l'écran
RIGHT$	R SHIFT	R
RND	R SHIFT N	R
RUN	R SHIFT U	R
SAVE	S SHIFT A	S ♠
SGN	S SHIFT G	S
SIN	S SHIFT I	S
SPC(	S SHIFT P	S
SQR	S SHIFT Q	S ●
STATUS	ST	ST
STEP	ST SHIFT E	ST
STOP	S SHIFT T	S
STR$	ST SHIFT R	ST
SYS	S SHIFT Y	S
TAB(	T SHIFT A	T ♠
TAN	Aucune	TAN
THEN	T SHIFT H	T
TIME	TI	TI
TIME$	TI$	TI$
USR	U SHIFT S	U ♥
VAL	V SHIFT A	V ♠
VERIFY	V SHIFT E	V
WAIT	W SHIFT A	W ♠

CODES D'AFFICHAGE D'ÉCRAN

Le tableau suivant donne tous les caractères des jeux intégrés au Commodore 64. Il indique les nombres à inscrire (POKE) dans la mémoire d'écran (positions 1024 à 2023) pour obtenir le caractère désiré. On y trouve aussi le caractère correspondant à un nombre lu (PEEK) de l'écran.

Sur les deux disponibles, on peut disposer d'un jeu de caractères à la fois. Sur l'écran, on ne peut donc pas afficher simultanément des caractères des deux jeux. On passe d'un jeu à l'autre en appuyant simultanément sur les touches SHIFT et C=

Du BASIC, POKE 53272,21 fait passer en majuscules et POKE 53272,23 fait passer en minuscules.

On peut aussi afficher tout nombre du tableau en vidéo inverse. Pour obtenir le code de caractère inverse, ajouter 128 aux valeurs indiquées.

Si l'on veut afficher un cercle plein à la position 1504, inscrire (POKE) le code du cercle (81) dans cette position: POKE 1504,81.

Une position correspondante de mémoire commande la couleur de chaque caractère affiché sur l'écran (positions 55296 à 56295). Pour faire passer la couleur du cercle au jaune (code de couleur 7), inscrire (POKE) dans la position de mémoire correspondante (55776) avec la couleur du caractère (POKE 55776,7).

Voir l'annexe D pour les topographies complètes de mémoire d'écran de couleur ainsi que les codes de couleur.

REMARQUE: Les instructions POKE suivantes affichent le même symbole dans les jeux 1 et 2: 1, 27-64, 91-93, 96-104, 106-121, 123-127.

CODES D'ÉCRAN

JEU 1	JEU 2	POKE	JEU 1	JEU 2	POKE	JEU 1	JEU 2	POKE
@		0	C	c	3	F	f	6
A	a	1	D	d	4	G	g	7
B	b	2	E	e	5	H	h	8

JEU 1	JEU 2	POKE	JEU 1	JEU 2	POKE	JEU 1	JEU 2	POKE
I	i	9	%		37	♠	A	65
J	j	10	&		38		B	66
K	k	11	'		39		C	67
L	l	12	(		40		D	68
M	m	13	)		41		E	69
N	n	14	*		42		F	70
O	o	15	+		43		G	71
P	p	16	,		44		H	72
Q	q	17	–		45		I	73
R	r	18	.		46		J	74
S	s	19	/		47		K	75
T	t	20	0		48		L	76
U	u	21	1		49		M	77
V	v	22	2		50		N	78
W	w	23	3		51		O	79
X	x	24	4		52		P	80
Y	y	25	5		53		Q	81
Z	z	26	6		54		R	82
[		27	7		55	♥	S	83
£		28	8		56		T	84
]		29	9		57		U	85
↑		30	:		58		V	86
←		31	;		59		W	87
SPACE		32	<		60	♣	X	88
!		33	=		61		Y	89
"		34	>		62	♦	Z	90
#		35	?		63			91
$		36			64			92

JEU 1	JEU 2	POKE	JEU 1	JEU 2	POKE	JEU 1	JEU 2	POKE
		93			105			117
		94			106			118
		95			107			119
SPACE		96			108			120
		97			109			121
		98			110		✓	122
		99			111			123
		100			112			124
		101			113			125
		102			114			126
		103			115			127
		104			116			

Les codes 128 à 255 sont les images inverses des codes 0 à 127

ANNEXE C

CODES ASCII et CHR$

Cet annexe indique les caractères obtenus quand on imprime (PRINT) CHR$(X), pour toutes les valeurs possibles de X. Il donne aussi les valeurs correspondantes quand on tape PRINT ASC("x") quand x correspond à tout caractère pouvant être tapé. Ce processus est utile pour reconnaître un caractère reçu dans une instruction GET, convertir les majuscules/minuscules et imprimer les commandes sous forme de caractères (comme le passage aux majuscules/minuscules) qui ne peuvent pas être placées entre guillemets.

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0	CRSR ↓	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
INVALIDE SHIFT C=	8		25	*	42	;	59
VALIDE SHIFT C=	9		26	+	43	<	60
	10		27	,	44	=	61
	11	RED	28	–	45	>	62
	12	CRSR →	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
SWITCH TO LOWER CASE	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
D	68		97		126	Gris 3	155
E	69		98		127	PUR	156
F	70		99		128	CRSR	157
G	71		100	Orange	129	YEL	158
H	72		101		130	CYN	159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	SWITCH TO UPPER CASE	142		171
U	85		114		143		172
V	86		115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120	Brun	149		178
£	92		121	Rouge clair	150		179
]	93		122	Gris 1	151		180
↑	94		123	Gris 2	152		181
←	95		124	Vert clair	153		182
	96		125	Bleu clair	154		183

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	184		186		188		190
	185		187		189		191

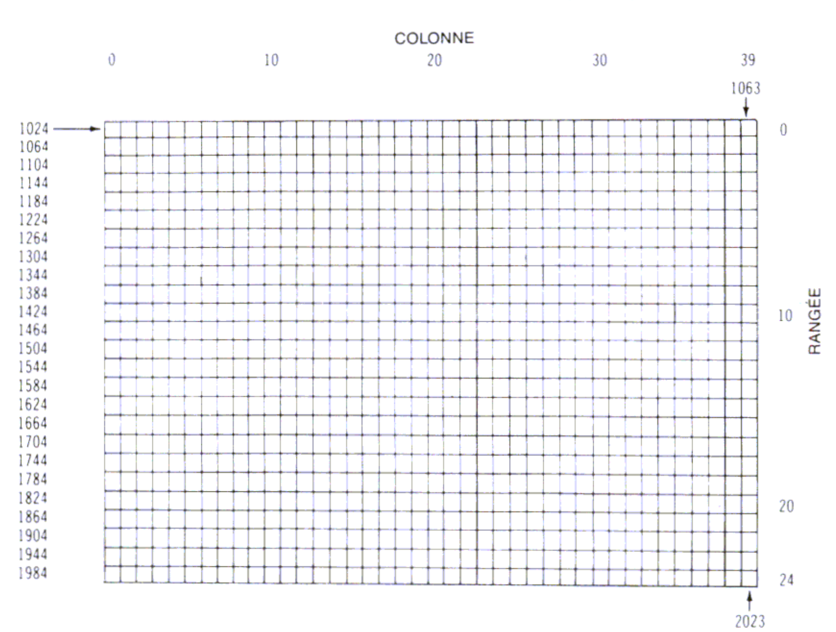
CODES	**192-223**	**IDENTIQUES A**	**96-127**
CODES	**224-254**	**IDENTIQUES A**	**160-190**
CODE	**255**	**IDENTIQUE A**	**126**

ANNEXE D

TOPOGRAPHIES DES MÉMOIRES D'ÉCRAN ET DE COULEUR

Les tableaux suivants indiquent les positions de mémoire qui commandent le placement des caractères sur l'écran et les positions utilisées pour changer les couleurs des caractères individuels. Ils indiquent aussi les codes de couleur des caractères.

TOPOGRAPHIE DE MÉMOIRE D'ÉCRAN

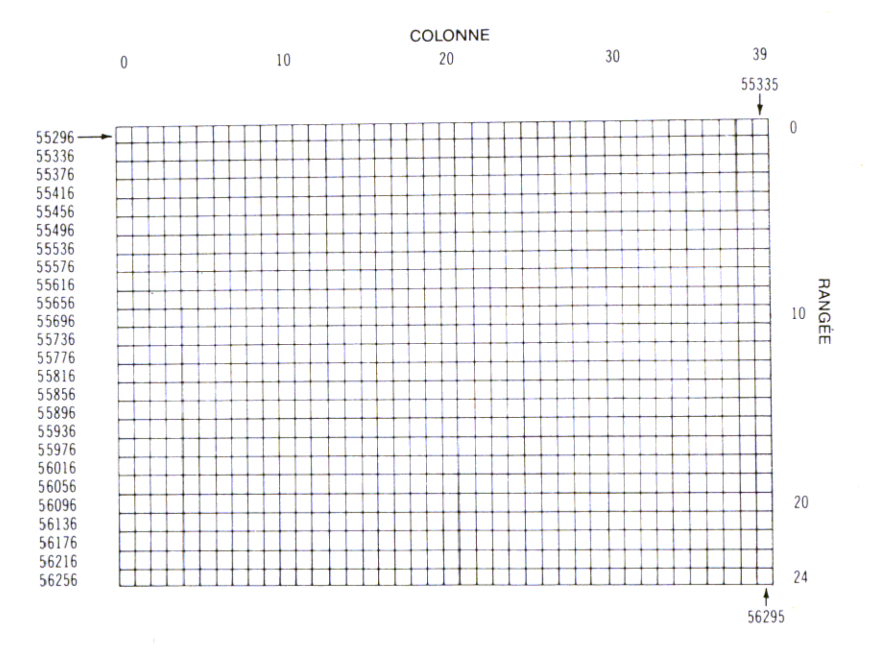


Pour changer une couleur de caractère, les valeurs réelles à inscrire (POKE) dans une position de mémoire sont:

0	NOIR	8	ORANGE
1	BLANC	9	BRUN
2	ROUGE	10	ROUGE CLAIR
3	TURQUOISE	11	GRIS 1
4	VIOLET	12	GRIS 2
5	VERT	13	VERT CLAIR
6	BLEU	14	BLEU CLAIR
7	JAUNE	15	GRIS 3

Par exemple, pour faire passer au rouge la couleur d'un caractère placé d ans le coin supérieur gauche de l'écran, taper: POKE 55296,2.

TOPOGRAPHIE DE MÉMOIRE DE COULEUR



ANNEXE E

VALEURS DES NOTES DE MUSIQUE

Cet annexe donne une liste complète des numéros de note, des notes et des valeurs à inscrire (POKE) dans les registres de haute fréquence et de basse fréquence de la microplaquette de son pour obtenir la note correspondante.

NOTE DE MUSIQUE		FRÉQUENCE D'OSCILLATEUR		
NOTE	OCTAVE	DÉCIMAL	HAUTE	BASSE
0	Do-0	268	1	12
1	Do#-0	284	1	28
2	Ré-0	301	1	45
3	Ré#-0	318	1	62
4	Mi-0	337	1	81
5	Fa-0	358	1	102
6	Fa#-0	379	1	123
7	Sol-0	401	1	145
8	Sol#-0	425	1	169
9	La-0	451	1	195
10	La#-0	477	1	221
11	Si-0	506	1	250
16	Do-1	536	2	24
17	Do#-1	568	2	56
18	Ré-1	602	2	90
19	Ré#-1	637	2	125
20	Mi-1	675	2	163
21	Fa-1	716	2	204
22	Fa#-1	758	2	246
23	Sol-1	803	3	35
24	Sol#-1	851	3	83
25	La-1	902	3	134
26	La#-1	955	3	187
27	Si-1	1012	3	244
32	Do-2	1072	4	48

NOTE DE MUSIQUE		FRÉQUENCE D'OSCILLATEUR		
NOTE	OCTAVE	DÉCIMAL	HAUTE	BASSE
33	Do#-2	1136	4	112
34	Ré-2	1204	4	180
35	Ré#-2	1275	4	251
36	Mi-2	1351	5	71
37	Fa-2	1432	5	152
38	Fa#-2	1517	5	237
39	Sol-2	1607	6	71
40	Sol#-2	1703	6	167
41	La-2	1804	7	12
42	La#-2	1911	7	119
43	Si-2	2025	7	233
48	Do-3	2145	8	97
49	Do#-3	2273	8	225
50	Ré-3	2408	9	104
51	Ré#-3	2551	9	247
52	Mi-3	2703	10	143
53	Fa-3	2864	11	48
54	Fa#-3	3034	11	218
55	Sol-3	3215	12	143
56	Sol#-3	3406	13	78
57	La-3	3608	14	24
58	La#-3	3823	14	239
59	Si-3	4050	15	210
64	Do-4	4291	16	195
65	Do#-4	4547	17	195
66	Ré-4	4817	18	209
67	Ré#-4	5103	19	239
68	Mi-4	5407	21	31
69	Fa-4	5728	22	96
70	Fa#-4	6069	23	181
71	Sol-4	6430	25	30
72	Sol#-4	6812	26	156
73	La-4	7217	28	49
74	La#-4	7647	29	223
75	Si-4	8101	31	165
80	Do-5	8583	33	135
81	Do#-5	9094	35	134

NOTE DE MUSIQUE		FRÉQUENCE D'OSCILLATEUR		
NOTE	OCTAVE	DÉCIMAL	HAUTE	BASSE
82	Ré-5	9634	37	162
83	Ré#-5	10207	39	223
84	Mi-5	10814	42	62
85	Fa-5	11457	44	193
86	Fa#-5	12139	47	107
87	Sol-5	12860	50	60
88	Sol#-5	13625	53	57
89	La-5	14435	56	99
90	La#-5	15294	59	190
91	Si-5	16203	63	75
96	Do-6	17167	67	15
97	Do#-6	18188	71	12
98	Ré-6	19269	75	69
99	Ré#-6	20415	79	191
100	Mi-6	21629	84	125
101	Fa-6	22915	89	131
102	Fa#-6	24278	94	214
103	Sol-6	25721	100	121
104	Sol#-6	27251	106	115
105	La-6	28871	112	199
106	La#-6	30588	119	124
107	Si-6	32407	126	151
112	Do-7	34334	134	30
113	Do#-7	36376	142	24
114	Ré-7	38539	150	139
115	Ré#-7	40830	159	126
116	Mi-7	43258	168	250
117	Fa-7	45830	179	6
118	Fa#-7	48556	189	172
119	Sol-7	51443	200	243
120	Sol#-7	54502	212	230
121	La-7	57743	225	143
122	La#-7	61176	238	248
123	Si-7	64814	253	46

RÉGLAGES DES FILTRES

Position	Fonction
54293	Fréquence de coupure basse (0-7)
54294	Fréquence de coupure haute (0-255)
54295	Résonnance (bits 4-7) Filtre de voix 3 (bit 2) Filtre de voix 2 (bit 1) Filtre de voix 1 (bit 0)
54296	Filtre passe-haut (bit 6) Filtre de bande passante (bit 5) Filtre passe-bas (bit 4) Volume (bits 0-3)

ANNEXE F

BIBLIOGRAPHIE

Addison-Wesley	"BASIC and the Personal Computer", Dwyer and Critchfield
Compute	"Compute's First Book of PET/CBM"
Cowbay Computing	"Feed Me, I'm Your PET Computer", Carol Alexander
	'Looking Good with Your PET", Carol Alexander
	"Teacher's PET — Plans, Quizzes, and Answers"
Creative Computing	"Getting Acquainted With Your VIC 20", T. Hartnell
Dilithium Press	"BASIC Basic-English Dictionary for the PET", Larry Noonan
	"PET BASIC", Tom Rugg et Phil Feldman
Éditions du P.S.I.	"La découverte du PET/CBM", par Daniel-Jean David
	"La pratique du PET/CBM", Vol. 1, par Daniel-Jean David
	"La pratique du PET/CBM", Vol. 2, par Daniel-Jean David
Faulk Baker Associates	"MOS Programming Manual", MOS Technology
Hayden Book Co.	"BASIC from the Ground Up", David E. Simon
	"I Speak BASIC to My PET", Aubrey Jones, Jr.
	"Library of PET Subroutines", Nick Hampshire
	"PET Graphics", Nick Hampshire
	"BASIC Conversions Handbook, Apple, TRS-80, and PET", David A. Brain, Phillip R. Oviatt, Paul J. Paquin et Chandler P. Stone

Howard W. Sams	"The Howard W. Sams Crash Course in Microcomputers", Louis E. Frenzel, Jr.
	"Mostly BASIC: Applications for Your PET", Howard Berenbon
	"PET Interfacing", James M. Downey et Steven M. Rogers
	"VIC 20 Programmer's Reference Guide", A. Finkel, P. Higginbottom, N. Harris et M. Tomczyk
Little, Brown & Co.	"Computer Games for Businesses, Schools, and Homes", J. Victor Nagigian et William S. Hodges
	"The Computer Tutor: Learning Activities for Homes and Schools", Gary W. Orwig, University of Central Florida, et William S. Hodges
McGraw-Hill	"Hands-On BASIC With a PET", Herbert D. Peckman
	"Home and Office Use of VisiCalc", D. Castlewitz et L. Chisauki
Osborne/McGraw-Hill	"PET/CBM Personal Computer Guide", Carroll S. Donahue
	"PET Fun and Games", R. Jeffries et G. Fisher
	"PET and the IEEE", A. Osborne et C. Donahue
	"Some Common BASIC Programs for the PET", L. Poole, M. Borchers et C. Donahue
	"Osborne CP/M User Guide", Thom Hogan
	"CBM Professional Computer Guide"
	"The PET Personal Guide"
	"The 8086 Book", Russell Rector and George Alexy
P.C. Publications	"Beginning Self-Teaching Computer Lessons"
Prentice-Hall	"The PET Personal Computer for Beginners", S. Dunn et V. Morgan

Reston Publishing Co.	"PET and the IEEE 488 Bus (GPIB)", Eugene Fisher and C. W. Jensen
	"PET BASIC—Training Your PET Computer", Ramon Zamora, Wm. F. Carrie et B. Allbrecht
	"PET Games and Recreation", M. Ogelsby, L. Lindsey et D. Kunkin
	"PET BASIC", Richard Huskell
	"VIC Games and Recreations"
Telmas Courseware Ratings	"BASIC and the Personal Computer", T. A. Dwyer et M. Critchfield
Total Information Services	"Understanding Your PET/CBM, Vol. 1, BASIC Programming"
	"Understanding Your VIC", David Schultz

Les magazines Commodore donnent les toutes dernières informations sur le Commodore 64. On peut sérieusement envisager de s'abonner aux deux publications suivantes qui sont très connues:

COMMODORE—*The Microcomputer Magazine*: publié tous les deux mois et disponible par abonnement ($15 par an aux États-Unis et $25 par an hors des États-Unis).

POWER/PLAY—*The Home Computer Magazine*: publié tous les trois mois et disponible par abonnement ($10 par an aux États-Unis et $15 hors des États-Unis.

ANNEXE G

TOPOGRAPHIE DES REGISTRES DE MICROPLAQUETTE VIC

Adresse de départ (base) 53248 ($D000)

n° de registre Déc.	Hex.	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7							S0X0	Car. graph. prog. 0, composante X
1	1	S0Y7							S0Y0	Car. graph. prog. 0, composante Y
2	2	S1X7							S1X0	Car. graph. prog. 1, X
3	3	S1Y7							S1Y0	Car. graph. prog. 1, Y
4	4	S2X7							S2X0	Car. graph. prog. 2, X
5	5	S2Y7							S2Y0	Car. graph. prog. 2, Y
6	6	S3X7							S3X0	Car. graph. prog. 3, X
7	7	S3Y7							S3Y0	Car. graph. prog. 3, Y
8	8	S4X7							S4X0	Car. graph. prog. 4, X
9	9	S4Y7							S4Y0	Car. graph. prog. 4, Y
10	A	S5X7							S5X0	Car. graph. prog. 5, X
11	B	S5Y7							S5Y0	Car. graph. prog. 5, Y
12	C	S6X7							S6X0	Car. graph. prog. 6, X
13	D	S6Y7							S6Y0	Car. graph. prog. 6, Y
14	E	S7X7							S7X0	Car. graph. prog. 7, composante X
15	F	S7Y7							S7Y0	Car. graph. prog. 7, composante Y
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	Bit le plus significatif de coord. X
17	11	RC8	ECM	BMM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	Mode de défilement Y
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Trame
19	13	LPX7							LPX0	Crayon lumineux X
20	14	LPY7							LPY0	Crayon lumineux Y

nº de registre Déc.	Hex.	DB7	DB6	DB5	DB4	DB3	DB2	DB1		DB0
21	15	SE7							SE0	Validation car. graph. prog. (Marche/arrêt)
22	16	N.C.	N.C.	RST	MCM	CSEL	XSCL2	XSCL1	XSCL0	Mode de défilement X
23	17	SEXY7							SEXY0	Car. graph. prog., allong. Y
24	18	VS13	VS12	VS11	VS10	CB13	CB12	CB11	N.C.	Écran et adresse de base de mémoire de caractère
25	19	IRQ	N.C.	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Demande d'interruption
26	1A	N.C.	N.C.	N.C.	N.C.	MLPI	MISSC	MISBC	MRIRQ	Masques de demande d'interruption
27	1B	BSP7							BSP0	Priorité arrière-plan/ car. graph. prog.
28	1C	SCM7							SCM0	Sélection car. graph. multicolore
29	1D	SEXX7							SEXX0	Car. graph. prog., allong. X
30	1E	SSC7							SSC0	Collision car. graph./ car. graph.
31	1F	SBC7							SBC0	Collision car. graph./ arrière-plan

n° de registre Déc.	Hex.	Couleur
32	20	Couleur de cadre
33	21	Arrière-plan, couleur 0
34	22	Arrière-plan, couleur 1
35	23	Arrière-plan, couleur 2
36	24	Arrière-plan, couleur 3
37	25	Car. graph. prog., couleur 0
38	26	Car. graph. prog., multicolore 1

n° de registre Déc.	Hex.	Couleur
39	27	Car. graph. prog., couleur 0
40	28	Car. graph. prog., couleur 1
41	29	Car. graph. prog., couleur 2
42	2A	Car. graph. prog., couleur 3
43	2B	Car. graph. prog., couleur 4
44	2C	Car. graph. prog., couleur 5
45	2D	Car. graph. prog., couleur 6
46	2E	Car. graph. prog., couleur 7

CODES DE COULEUR

Déc.	Hex.	Couleur
0	0	NOIR
1	1	BLANC
2	2	ROUGE
3	3	TURQUOISE
4	4	VIOLET
5	5	VERT
6	6	BLEU
7	7	JAUNE

Déc.	Hex.	Couleur
8	8	ORANGE
9	9	BRUN
10	A	ROUGE CLAIR
11	B	GRIS 1
12	C	GRIS 2
13	D	VERT CLAIR
14	E	BLEU CLAIR
15	F	GRIS 3

LÉGENDE:
ON NE PEUT UTILISER QUE LES COULEURS 0 à 7 EN MODE MULTICOLORE DE CARACTÈRES.

ANNEXE H

CALCUL DES FONCTIONS MATHÉMATIQUES

Le tableau suivant permet de calculer les fonctions qui ne sont pas intrinsèques au BASIC de Commodore 64:

FONCTION	ÉQUIVALENT BASIC
SÉCANTE	SEC(X)=1/COS(X)
COSÉCANTE	CSC(X)=1/SIN(X)
COTANGENTE	COT(X)=1/TAN(X)
SINUS INVERSE	ARCSIN(X)=ATN(X/SQR(-X*X+1))
COSINUS INVERSE	ARCCOS(X)=ATN(X/SQR (-X*X+1)) + π/2
SÉCANTE INVERSE	ARCSEC(X)=ATN(X/SQR(X*X-1))
COSÉCANTE INVERSE	ARCCSC(X)=ATN(X/SQR(X*X-1)) +(SGN(X)-1*π/2
COTANGENTE INVERSE	ARCOT(X)=ATN(X)+π/2
SINUS HYPERBOLIQUE	SINH(X)=(EXP(X)-EXP(-X))/2
COSINUS HYPERBOLIQUE	COSH(X)=(EXP(X)+EXP(-X))/2
TANGENTE HYPERBOLIQUE	TANH(X)=EXP(-X)/(EXP(X)+EXP (-X))*2+1
SÉCANTE HYPERBOLIQUE	SECH(X)=2/(EXP(X)+EXP(-X))
COSÉCANTE HYPERBOLIQUE	CSCH(X)=2/(EXP(X)-EXP(-X))
COTANGENTE HYPERBOLIQUE	COTH(X)=EXP(-X)/(EXP(X) -EXP(-X))*2+1
SINUS HYPERBOLIQUE INVERSE	ARCSINH(X)=LOG(X+SQR(X*X+1))
COSINUS HYPERBOLIQUE INVERSE	ARCCOSH(X)=LOG(X+SQR(X*X-1))
TANGENTE HYPERBOLIQUE INVERSE	ARCTANH(X)=LOG(1+X)/(1-X))/2
SÉCANTE HYPERBOLIQUE INVERSE	ARCSECH(X)=LOG((SQR (-X*X+1)+1/X)
COSÉCANTE HYPERBOLIQUE INVERSE	ARCCSCH(X)=LOG((SGN(X)*SQR (X*X+1/x)
COTANGENTE HYPERBOLIQUE INVERSE	ARCCOTH(X)=LOG((X+1)/(X-1))/2

ANNEXE I

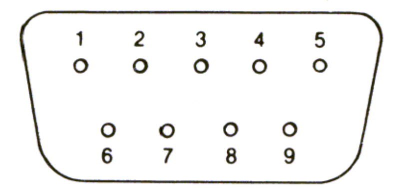
BROCHAGES DES DISPOSITIFS D'ENTRÉE/SORTIE

Cet annexe indique les branchements pouvant être fait avec le Commodore 64.

1) Entrée/sortie jeu
2) Fente de cartouche
3) Son/vidéo
4) Entrée/sortie série (disque/imprimante)
5) Sortie modulateur
6) Cassette
7) Accès d'utilisateur

Accès de commande 1

Broche	Type	Remarque
1	MANETTE A0	
2	MANETTE A1	
3	MANETTE A2	
4	MANETTE A3	
5	POT. AY	
6	BOUTON A/LP	
7	+ 5 V	50 mA MAX.
8	MASSE	
9	POT. AX	



Accès de commande 2

Broche	Type	Remarque
1	MANETTE B0	
2	MANETTE B1	
3	MANETTE B2	
4	MANETTE B3	
5	POT. BY	
6	BOUTON B	
7	+ 5 V	50 mA MAX.
8	MASSE	
9	POT. BX	

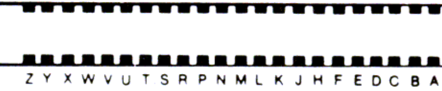
Fente d'extension de cartouche

Broche	Type
1	MASSE
2	+ 5 V
3	+ 5 V
4	IRQ
5	R/W
6	HORLOGE DE POINT
7	ENTRÉE/SORTIE 1
8	JEU
9	EX ROM
10	ENTRÉE/SORTIE 2
11	ROML

Broche	Type
12	BA
13	DMA
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	MASSE

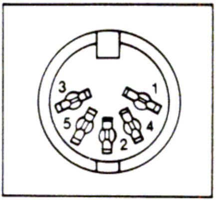
Broche	Type
A	MASSE
B	ROMH
C	REMISE ÉTAT INITIAL
D	NMI
E	S 02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10

Broche	Type
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	MASSE



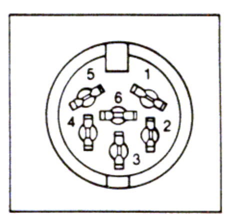
Son/vidéo

Broche	Type	Remarque
1	LUMINANCE	
2	MASSE	
3	SORTIE SON	
4	SORTIE VIDÉO	
5	ENTRÉE SON	



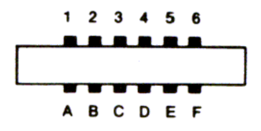
Entrée/sortie série

Broche	Type
1	SRQIN SÉRIE
2	MASSE
3	ENTRÉE/SORTIE ATN SÉRIE
4	ENTRÉE/SORTIE HORLOGE SÉRIE
5	ENTRÉE/SORTIE DONNÉES SÉRIE
6	REMISE À L'ÉTAT INITIAL



Cassette

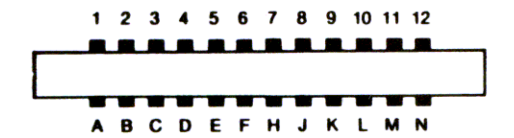
Broche	Type
A-1	MASSE
B-2	+5V
C-3	MOTEUR DE MAGNÉTOCASSETTE
D-4	LECTURE CASSETTE
E-5	ÉCRIRE CASSETTE
F-6	DÉTECTION CASSETTE



Entrée/sortie utilisateur

Broche	Type	Remarque
1	MASSE	
2	+5V	100 mA MAX.
3	REMISE À L'ÉTAT INITIAL	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	PC2	
9	ENTRÉE ATN SÉRIE	
10	CA 9 V	100 mA MAX.
11	CA 9 V	100 mA MAX.
12	MASSE	

Broche	Type	Remarque
A	MASSE	
B	INDICATEUR 2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PB8	
N	MASSE	



ANNEXE J

CONVERSION DES PROGRAMMES DE BASIC STANDARD EN BASIC DE COMMODORE 64

Si l'on possède des programmes écrits dans un BASIC autre que celui du Commodore, il peut falloir leur apporter quelques modifications mineures avant de pouvoir les exécuter avec le Commodore 64. Dans cette annexe, nous donnons quelques conseils pour faciliter la conversion.

Dimensions des chaînes

Supprimer toutes les instructions utilisées pour déclarer la longueur des chaînes. Une instruction comme DIM A$(I,J), qui dimensionne un tableau de chaînes pour J éléments de longueur I, doit être convertie à l'instruction BASIC de Commodore DIM A$(J).

Dans certains BASIC, on utilise une virgule ou un luète (signe &) pour la concaténation des chaînes. Chacun de ces signes doit être remplacé par un signe plus qui correspond à l'opérateur BASIC de Commodore pour la concaténation des chaînes.

En BASIC de Commodore 64, les fonction MID$, RIGHT$ et LEFT$ servent à tirer des sous-chaînes de chaînes. Les formes du genre A$(I), pour accéder au I^{e} caractère dans A$, ou A$(I,J), pour tirer une sous-chaîne de A$ de la position I à la position J, doivent être remplacées par:

Autre BASIC	BASIC de Commodore 64
A$(I) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,I+1)
A$(I,J) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,J+1)

Attributions multiples

Pour fixer B et C égaux à zéro, certains BASIC permettent des instructions du genre:

```
10 LET B=C=0
```

Le BASIC de Commodore 64 interprète le deuxième signe d'égalité comme un opérateur logique et donne B = −1 si C = 0. Convertir cette instruction pour qu'on lise:

```
10 C=0 : B=0
```

Instructions multiples

Dans certains BASIC, on utilise une oblique inverse (/) pour séparer les instructions multiples dans une ligne. Avec le BASIC de Commodore 64, on sépare toutes les instructions par un deux-points (:).

Fonctions MAT

Les programmes utilisant les fonctions MAT, disponibles dans certains BASIC, doivent être réécrits avec des boucles FOR . . . NEXT pour arriver à une exécution convenable.

ANNEXE K

MESSAGES D'ERREUR

Cette annexe donne une liste complète des messages d'erreur créés par le Commodore 64, avec la description de leurs causes.

BAD DATA (données erronées) Des données en chaîne ont été reçues d'un fichier ouvert, mais le programme attendait des données numériques.

BAD SUBSCRIPT (indice inférieur erroné) Le programme essayait de désigner un élément d'un tableau dont le numéro était à l'extérieur de l'intervalle spécifié dans l'instruction DIM.

BREAK (interruption) L'exécution du programme s'est arrêtée parce que l'on a appuyé sur la touche STOP .

CAN'T CONTINUE (impossible de continuer) La commande CONT ne fonctionne pas, soit parce que le programme n'a jamais été exécuté (RUN), soit parce qu'il y a eu une erreur ou parce qu'une ligne a été corrigée.

DEVICE NOT PRESENT (absence du dispositif) Le dispositif d'entrée/sortie requis n'était pas disponible pour une instruction OPEN, CLOSE, CMD, PRINT#, INPUT# ou GET#.

DIVISION BY ZERO (division par zéro) La division par zéro est une impossibilité mathématique et, de ce fait, n'est pas permise.

EXTRA IGNORED (éléments supplémentaires omis) Un trop grand nombre d'éléments de données ont été tapés en réponse à une instruction INPUT. Seuls les premiers éléments ont été acceptés.

FILE NOT FOUND (fichier non trouvé) Si l'on cherchait un fichier sur cassette, une marque de fin de bande (END-OF-TAPE) a été localisée. Si l'on cherchait un fichier sur disque, aucun fichier répondant à ce nom n'existait.

FILE NOT OPEN (fichier non ouvert) Le fichier spécifié dans une instruction CLOSE, CMD, PRINT#, INPUT# OU GET# doit d'abord être ouvert (OPEN).

FILE OPEN (fichier ouvert) On a essayé d'ouvrir un fichier en utilisant le numéro d'un fichier déjà ouvert.

FORMULA TOO COMPLEX (formule trop complexe) L'expression en chaîne à évaluer doit être divisée en deux parties au moins pour que le système puisse travailler; il se peut aussi qu'une formule contienne trop de parenthèses.

ILLEGAL DIRECT (mode direct non autorisé) On ne peut utiliser l'instruction INPUT que dans un programme et non en mode direct.

ILLEGAL QUANTITY (quantité non autorisée) Un nombre utilisé comme argument d'une fonction ou instruction est en dehors de l'intervalle permis.

LOAD (chargement) Il se présente un problème avec le programme sur bande.

NEXT WITHOUT FOR (NEXT sans FOR) Erreur causée par des boucles incorrectement emboîtées ou ayant un nom de variable dans une instruction NEXT ne correspondant pas à un autre dans une instruction FOR.

NOT INPUT FILE (fichier non prévu pour l'entrée) On a essayé d'introduire (INPUT) des données dans un fichier spécifié pour la sortie uniquement ou d'en tirer (GET) des données.

NOT OUTPUT FILE (fichier non prévu pour la sortie) On a essayé d'imprimer (PRINT) des données dans un fichier spécifié pour l'entrée seulement.

OUT OF DATA (fin des données) On a exécuté une instruction READ, mais il ne reste plus de données à lire dans l'instruction DATA.

OUT OF MEMORY (fin de mémoire) Il n'y a plus de mémoire vive RAM disponible pour le programme ou les variables. Ce problème peut aussi se produire quand on a emboîté trop de boucles FOR ou quand il y a trop d'instructions GOSUB en fonction.

OVERFLOW (dépassement de capacité) Le résultat d'un calcul est supérieur au plus grand nombre permis, qui est 1.70141884E+38.

REDIM'D ARRAY (redimensionnement d'un tableau) On ne peut dimensionner (DIM) un tableau qu'une seule fois. Si l'on utilise une variable de tableau avant que le tableau en cause soit dimensionné, une opération DIM automatique est exécutée sur ce tableau pour fixer le nombre d'éléments à 10; toute opération DIM ultérieure amène cette erreur.

REDO FROM START (refaire depuis le début) On a tapé des données de caractère pendant une instruction INPUT alors que la machine attendait des données numériques. Il suffit de retaper l'introduction correctement pour que le programme continue de lui-même.

RETURN WITHOUT GOSUB (RETURN sans GOSUB) Le programme a rencontré une instruction RETURN alors qu'aucune commande GOSUB n'a été présentée.

STRING TOO LONG (chaîne trop longue) Une chaîne peut contenir un maximum de 255 caractères.

?SYNTAX ERROR (?erreur de syntaxe) Le Commodore 64 ne reconnaît pas l'instruction. Il peut manquer des parenthèses; des mots clés peuvent être mal orthographiés, etc.

TYPE MISMATCH (erreur d'assortiment de caractères) Cette erreur se produit quand on utilise un nombre à la place d'une chaîne ou vice versa.

UNDEF'D FUNCTION (fonction non définie) Une fonction définie par l'utilisateur a été désignée, mais elle n'a jamais été définie avec l'instruction DEF FN.

UNDEF'D STATEMENT (instructino non définie) Il y a eu un renvoi (GOTO ou GOSUB) ou une exécution (RUN) concernant un numéro de ligne qui n'existe pas.

VERIFY (vérifier) Le programme sur cassette ou sur disque ne correspond pas au programme présentement en mémoire.

ANNEXE L

SPÉCIFICATIONS DU MICROPROCESSEUR 6510

DESCRIPTION

Le 6510 est un système de micro-ordinateur économique capable de résoudre à bon marché une grande diversité de problèmes pour petits systèmes et périphériques.

Un accès d'entrée/sortie bidirectionnel de 8 bits se trouve sur la microplaquette, avec le registre de sortie à l'adresse 0000 et le registre de direction de données à l'adresse 0001. L'accès d'entrée/sortie est programmable bit par bit.

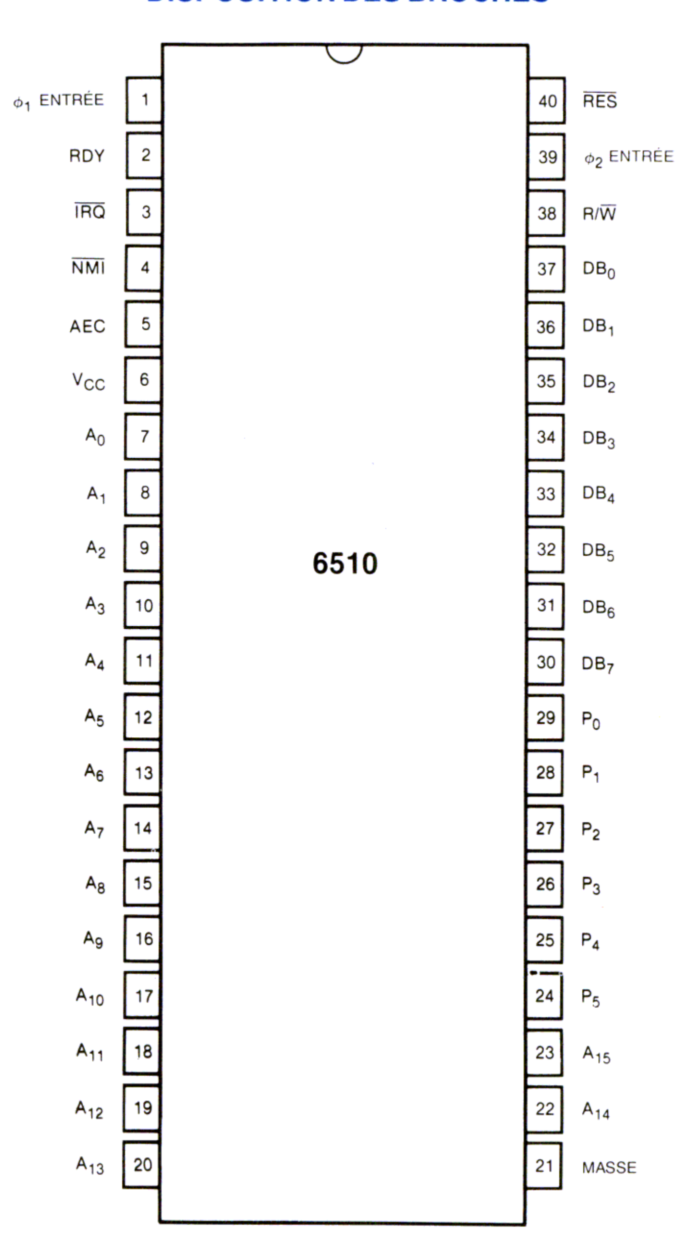
Le bus d'adresse à seize bits à trois états permet l'utilisation des systèmes à accès direct à la mémoire et à multiprocesseur partageant une mémoire commune.

La structure interne de l'unité de traitement est identique à celle du 6502 à technologie MOS (semi-conducteur métal-oxyde) afin de permettre la compatibilité des logiciels.

CARACTÉRISTIQUES DU 6510 . . .

- Accès d'entrée/sortie bidirectionnel huit bits
- Alimentation simple +5 volts
- Technologie des charges appauvries, porte au silicium, canal N
- Traitement parallèle huit bits
- 56 instructions
- Arithmétique décimale et binaire
- Treize modes d'adressage
- Possibilité d'indexation vraie
- Pointeur de pile programmable
- Pile de longueur variable
- Possibilité d'interruption
- Bus de données bidirectionnel huit bits
- Intervalle de mémoire adressable atteignant 65 K-octets
- Possibilité d'accès direct à la mémoire
- Compatible par bus avec le M6800
- Structure pipeline
- Exploitation sur 1 et 2 MHz
- S'utilise avec tout type de mémoire rapide

DISPOSITION DES BROCHES



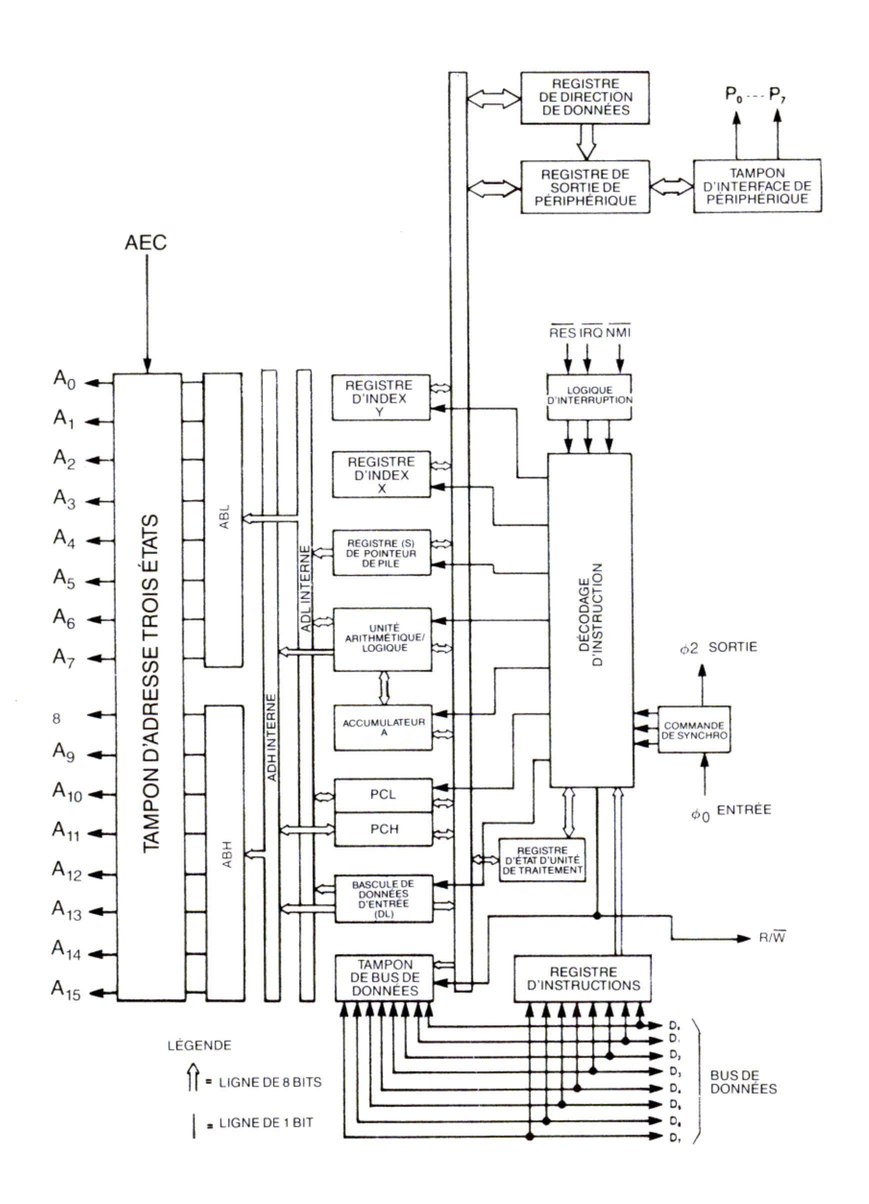


SCHÉMA DE PRINCIPE DU 6510

CARACTÉRISTIQUES DU 6510

VALEURS MAXIMALES

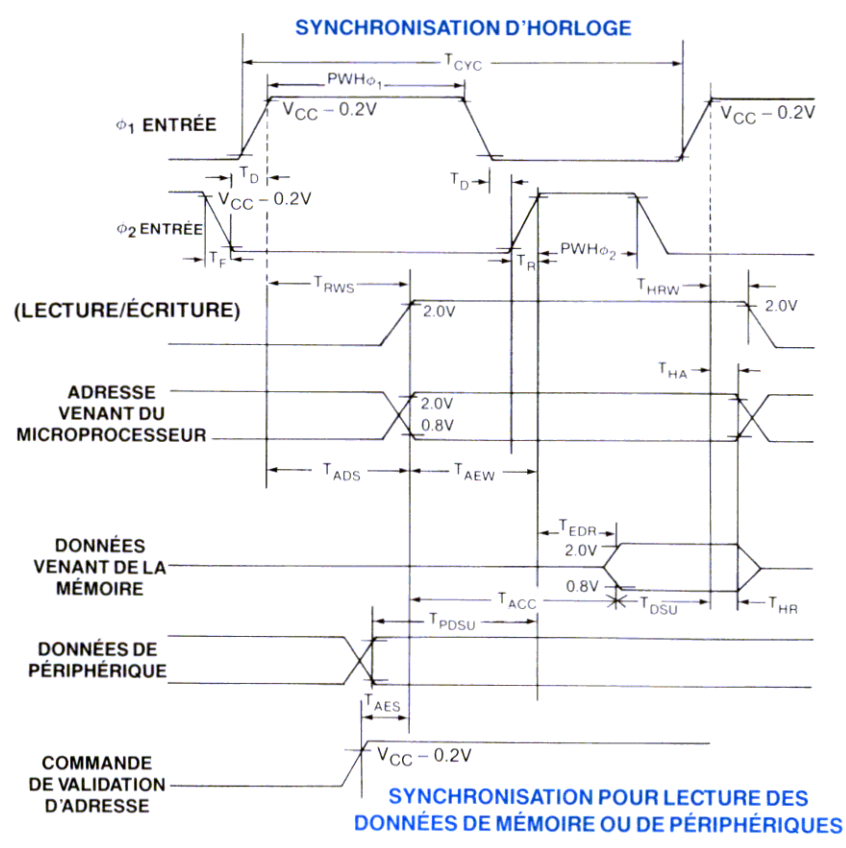
VALEUR	SYMBOLE	INTERVALLE	UNITÉ
TENSION D'ALIMENTATION	V_{CC}	−0.3 à +7.0	V_{CC}
TENSION D'ENTRÉE	V_{in}	−0.3 à +7.0	V_{CC}
TEMPÉRATURE DE FONCTIONNEMENT	T_A	0 à +70	°C
TEMPÉRATURE DE REMISAGE	T_{STG}	−55 à +150	°C

REMARQUE: Ce dispositif comprend une protection d'entrée contre les dommages dus aux tensions statiques ou aux champs électriques élevés; toutefois, prendre les précautions nécessaires pour éviter d'appliquer des tensions supérieures à la valeur maximale indiquée.

CARACTÉRISTIQUES ÉLECTRIQUES (V_{CC} = 5.0 V ±5%, V_{SS} = 0, T_A = 0° à +70°C)

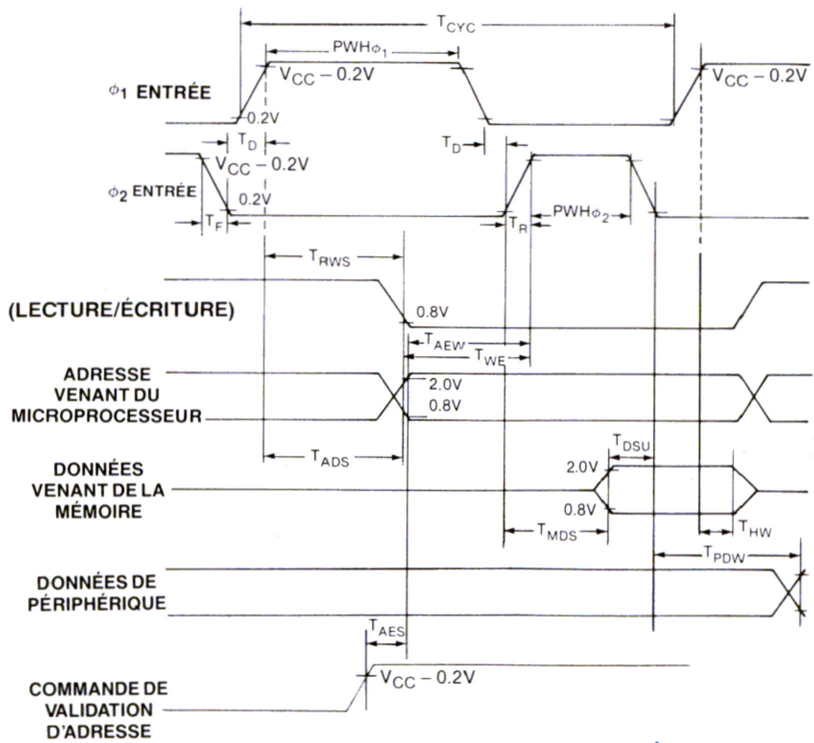
CARACTÉRISTIQUE	SYMBOLE	MIN.	TYP.	MAX.	UNITÉ
Tension haute d'entrée $\phi_1, \phi_{2(entrée)}$	V_{IH}	V_{CC} − 0.2	—	V_{CC} + 1.0V	V_{CC}
Tension haute d'entrée $\overline{RES}$, P_0-P_7 $\overline{IRQ}$, données		V_{SS} + 2.0	—	—	V_{CC}
Tension basse d'entrée $\phi_1, \phi_{2(entrée)}$	V_{IL}	V_{SS} − 0.3	—	V_{SS} + 0.8	V_{CC}
$\overline{RES}$, P_0-P_7 $\overline{IRQ}$, données		—	—	V_{SS} + 0.8	V_{CC}
Courant de fuite d'entrée (V_{in} = 0 à 5.25 V, V_{CC} = 5.25 V) Logique	I_{in}	—	—	2.5	μA
$\phi_1, \phi_{2(entrée)}$		—	—	100	μA
Courant d'entrée trois états (à l'arrêt) (V_{in} = 0.4 à 2.4 V, V_{CC} = 5.25 V) Lignes de données	I_{TSI}	—	—	10	μA
Tension haute de sortie (I_{OH} = −100 μA_{CC}, V_{CC} = 4.75 V) Données, A0-A15, R/W, P_0-P_7	V_{OH}	V_{SS} + 2.4	—	—	V_{CC}

CARACTÉRISTIQUE	SYMBOLE	MIN.	TYP.	MAX.	UNITÉ
Tension basse de sortie (I_{OL} = 1.6 mA_{CC}, V_{CC} = 4.75 V) Données, A0-A15, R/W, P_0-P_7	V_{OL}	—	—	V_{SS} + 0.4	V_{CC}
Courant d'alimentation	I_{CC}	—	125		mA
Capacité V_{in} = 0, T_A = 25°C, f = MHz) Logique, P_0-P_7	C_{in}	—	—	10	
Données		—	—	15	
A0-A15, R/W	C_{out}	—	—	12	
ϕ_1	$C\phi_1$	—	30	50	
ϕ_2	$C\phi_2$	—	50	80	



SYNCHRONISATION POUR LECTURE DES DONNÉES DE MÉMOIRE OU DE PÉRIPHÉRIQUES

SYNCHRONISATION D'HORLOGE



SYNCHRONISATION POUR ÉCRITURE DES DONNÉES VERS MÉMOIRE OU PÉRIPHÉRIQUES

CARACTÉRISTIQUES CA

CARACTÉRISTIQUES ÉLECTRIQUES(V_{CC} = 5 V ±5%, V_{SS} = 0 V, T_A = 0°—70°C)

SYNCHRO D'HORLOGE		SYNCHRO 1 MHz			SYNCHRO 2 MHz			
CARACTÉRISTIQUE	SYMBOLE	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	UNITÉ
Durée de cycle	T_{CYC}	1000	—	—	500	—	—	ns
Largeur d'impulsion d'horloge $\phi 1$ (Mesurée à V_{CC} — 0.2 V) $\phi 2$	PWHϕ1 PWHϕ	430 470	— —	— —	215 235	— —	— —	ns ns
Temps de chute, temps de montée (Mesuré de 0.2 V à V_{CC} — 0.2 V)	T_F, T_R	—	—	25	—	15	ns	
Retard entre horloges (Mesuré à 0.2 V)	T_D	0	—	—	0	—	—	ns

SYNCHRO DE LECTURE/ÉCRITURE (CHARGE = 1TTL)

		SYNCHRO 1 MHz			SYNCHRO 2 MHz			
CARACTÉRISTIQUE	SYMBOLE	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	UNITÉ
Temps d'établissement de lecture/écriture de 6508	T_{RWS}	—100	300	—	100	150	ns	
Temps d'établissement d'adresse de 6508	T_{ADS}	—	100	300	—	100	150	ns
Temps d'accès de lecture mémoire	T_{ACC}	—	—	575	—	—	300	ns

Période de stabilité de données	T_{DSU}	100	—	—	50			ns
Temps de maintien des données—lecture	T_{HR}		—	—				ns
Temps de maintien des données—écriture	T_{HW}	10	30	—	10	30		ns
Temps d'établissement des données de 6510	T_{MDS}	—	150	200	—	75	100	ns
Temps de maintien d'adresse	T_{HA}	10	30	—	10	30		ns
Temps de maintien de lecture/écriture	T_{HRW}	10	30	—	10	30		ns
Retard, adresse valide à la transition positive $\phi2$	T_{AEW}	180	—	—				ns
Retard, transition positive ϕ aux données valides sur bus	T_{EDR}	—	—	395				ns
Retard, données valides à la transition négative $\phi2$	T_{DSU}	300	—	—				ns
Retard, transition négative lecture/écriture à la transition positive ϕ	T_{WE}	130	—	—				ns
Retard, transition négative $\phi2$ aux données valides de périphérique	T_{PDW}	—	—	1				μs
Temps d'établissement de données de périphérique	T_{PDSU}	300	—	—				ns
Temps d'établissement de validation d'adresse	T_{AES}			60			60	ns

DESCRIPTION DES SIGNAUX

Horloges (ϕ_1, ϕ_2)

Le 6510 doit être équipé d'une horloge biphasée sans chevauchement, fonctionnant au niveau de tension V_{CC}.

Bus d'adresse (A_0-A_{15})

Ces sorties compatibles TTL peuvent commander une charge TTL normale et 130 pF.

Bus de données (D_0-D_7)

Huit broches sont attribuées à ce bus bidirectionnel qui assure le transfert des données entre le dispositif et les périphériques. Les sorties sont des tampons à trois états capables de commander une charge TTL normale et 130 pF.

Remise à l'état initial

Cette entrée sert à remettre le microprocesseur à l'état initial ou à le remettre en marche après une coupure d'alimentation. Quand cette ligne est à l'état bas, l'écriture à partir ou en direct du microprocesseur est bloquée. À la détection d'une impulsion positive à l'entrée, le microprocesseur commence immédiatement la remise à l'état initial.

Après une initialisation du système de six cycles d'horloge, l'indicateur d'interruption de masque est fixé et le microprocesseur charge le compteur de programme à partir des positions de vecteur de mémoire FFFC et FFFD. On est maintenant à la position de départ de commande de programme.

Quand V_{CC} atteint 4.75 volts pendant la mise sous tension, la remise à l'état initial doit être maintenue à l'état bas, pendant deux cycles d'horloge au moins. À ce moment, le signal de lecture/écriture devient valide.

Quand le signal de remise à l'état initial passe à l'état haut après ces deux cycles d'horloge, le microprocesseur passe à l'opération normale de remise à l'état initial décrite ci-dessus.

Demande d'interruption ($\overline{IRQ}$)

Cette entrée de niveau TTL demande le début d'une séquence d'interruption dans le microprocesseur. Le microprocesseur termine l'instruction présentement exécutée avant d'accepter la demande. À ce moment, le bit de masque d'interruption, dans le registre de codes d'état, est examiné. Si l'indicateur de masque d'interruption n'est pas fixé, le microprocesseur commence une séquence d'interruption. Le compteur de programme et le registre d'état d'unité de traitement sont stockés dans la pile. Le microprocesseur fixe alors l'indicateur de masque d'interruption à l'état haut pour qu'il ne se produise plus aucune interruption. À la fin de ce cycle, le compteur bas de programme se charge à partir de l'adresse FFFE et le compteur haut de programme à partir de l'adresse FFFF; la commande de programme est ainsi transférée au lecteur de mémoire situé à ces adresses.

Commande de validation d'adresse (AEC)

Le bus d'adresse n'est valide que si la ligne de commande de validation d'adresse est à l'état haut. À l'état bas, le bus d'adresse présente une impédance élevée. Cette caractéristique est pratique pour les systèmes à multiprocesseur et à accès direct à la mémoire.

Accès d'entrée/sortie (P_0-P_7)

Six broches équipent l'accès de périphérique qui assure le transfert des données à partir ou en direction des dispositifs périphériques. Le registre de sortie se trouve en mémoire vive à l'adresse 0001 et le registre de direction de données, à l'adresse 0000. Les sorties peuvent commander une charge TTL normale et 130 pF.

Lecture/écriture (R/W)

Ce signal, créé par le microprocesseur, commande la direction des transferts de données sur le bus de données. Cette ligne est à l'état haut, sauf quand le microprocesseur écrit vers la mémoire ou un dispositif périphérique.

MODES D'ADRESSAGE

ADRESSAGE D'ACCUMULATEUR—Ce type d'adressage est représenté par une instruction d'un octet qui implique une opération dans l'accumulateur.

ADRESSAGE IMMÉDIAT—En adressage immédiat, le facteur est contenu dans le deuxième octet de l'instruction et aucun autre adressage de mémoire n'est requis.

ADRESSAGE ABSOLU—En adressage absolu, le deuxième octet de l'instruction précise les huit bits de poids faible de l'adresse réelle; le troisième octet spécifie les huit bits de poids fort. Le mode d'adressage absolu permet ainsi l'accès à la totalité des 65 K-octets de la mémoire adressable.

ADRESSAGE DE PAGE ZÉRO—Les instructions de page zéro permettent un code et des temps d'exécution plus courts en n'extrayant que le deuxième octet de l'instruction et en supposant un octet zéro d'adresse haute. L'utilisation réfléchie de la page zéro peut se traduire par une efficacité nettement accrue des codes.

ADRESSAGE DE PAGE ZÉRO INDEXÉE—(Indexation X, Y)—Ce type d'adressage s'utilise avec le registre d'index; il est dit "Page zéro, X" ou "Page zéro, Y". L'adresse réelle se calcule en additionnant le deuxième octet au contenu du registre d'index. Avec cet adressage "Page zéro", le contenu du deuxième octet se rapporte à une position dans la page zéro. En outre, du fait de la fonction d'adressage de "Page zéro" de ce mode, il n'est ajouté aucun report aux 8 bits de poids fort de mémoire et il ne se produit pas de passage des limites de page.

ADRESSAGE ABSOLU INDEXÉ—(Indexation X, Y)—Ce type d'adressage s'utilise avec les registres d'index X et Y; il est dit "Absolu, X", et "Absolu, Y". L'adresse réelle se calcule en additionnant les contenus de X et Y à l'adresse contenue dans les deuxième et troisième octets de l'instruction. Ce mode permet au registre d'index de contenir l'index ou valeur de compte. L'instruction contient l'adresse de base. Ce type d'indexation permet l'affectation de positions. L'index peut modifier les zones multiples et conduire à un codage et un temps d'exécution réduits.

ADRESSAGE IMPLICITE—En mode d'adressage implicite, l'adresse contenant le facteur est implicitement énoncée dans le code d'opération de l'instruction.

ADRESSAGE RELATIF—L'adressage relatif ne s'utilise qu'avec les instructions de branchement; il établit une destination pour le branchement conditionnel.

Le deuxième octet de l'instruction devient le facteur qui correspond à un "décalage" ajouté au contenu des huit bits de poids faible du compteur de programme quand celui-ci est fixé à l'instruction suivante. L'intervalle de décalage s'étend de −128 à +127 octets de l'instruction suivante.

ADRESSAGE INDIRECT INDEXÉ—En adressage indirect indexé (Indirect, X), le deuxième octet de l'instruction est ajouté au contenu du registre d'index X et élimine le report. Le résultat de cette addition pointe vers une position de mémoire de la page zéro dont le contenu correspond aux huit bits de poids faible de l'adresse réelle. La position suivante de mémoire dans la page zéro contient les huit bits de poids fort de l'adresse réelle. Les deux positions de mémoire précisant les octets de poids fort et de poids faible de l'adresse réelle doivent être dans la page zéro.

ADRESSAGE INDEXÉ INDIRECT—En adressage indexé indirect [(Indirect), Y)], le deuxième octet de l'instruction pointe vers une position de mémoire dans la page zéro. Le contenu de cette position de mémoire est ajouté à celui du registre d'index Y; le résultat donne les huit bits de poids faible de l'adresse réelle. Le report de cette addition est ajouté au contenu de la position suivante de mémoire de page zéro; le résultat donne les huit bits de poids fort de l'adresse réelle.

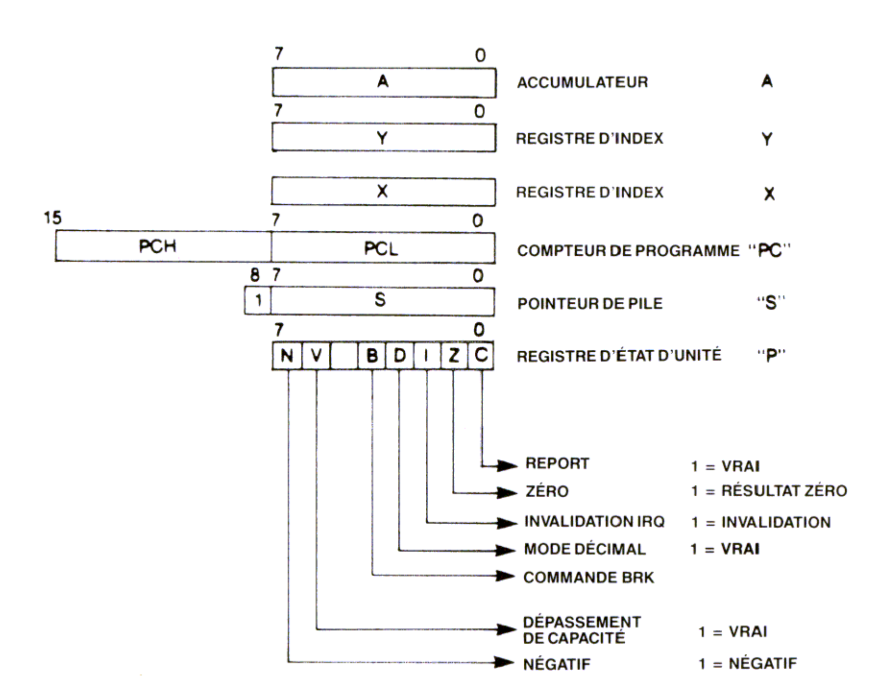
INDIRECT ABSOLU—Le deuxième octet de l'instruction contient les huit bits de poids faible d'une position de mémoire. Les huit bits de poids fort de cette position se trouvent dans le troisième octet de l'instruction. Le contenu de la position de mémoire spécifiée correspond à l'octet de poids faible de l'adresse réelle. La position suivante de mémoire contient l'octet de poids fort de l'adresse réelle qui est chargée dans les seize bits du compteur de programme.

JEU D'INSTRUCTIONS PAR ORDRE ALPHABÉTIQUE

ADC	Addition de la mémoire à l'accumulateur avec report
AND	Opération "ET" de la mémoire avec l'accumulateur
ASL	Décalage d'un bit vers la gauche (mémoire ou accumulateur)
BCC	Branchement sur effacement du report
BCS	Branchement sur mise du report à un
BEQ	Branchement sur le résultat zéro
BIT	Vérification des bits en mémoire avec l'accumulateur
BMI	Branchement sur résultat négatif
BNE	Branchement sur résultat non nul
BPL	Branchement sur résultat positif
BRK	Interruption forcée
BVC	Branchement sur effacement de dépassement de capacité
BVS	Branchement sur mise à un du dépassement de capacité
CLC	Effacement de l'indicateur de report
CLD	Effacement du mode décimal
CLI	Effacement du bit d'invalidation d'interruption
CLV	Effacement de l'indicateur de dépassement de capacité
CMP	Comparaison de la mémoire et de l'accumulateur
CPX	Comparaison de la mémoire et de l'index X
CPY	Comparaison de la mémoire et de l'index Y
DEC	Régression de un de la mémoire
DEX	Régression de un de l'index X
DEY	Régression de un de l'index Y
EOR	Opération "OU exclusif" de la mémoire avec l'accumulateur
INC	Progression de un de la mémoire
INX	Progression de un de l'index X
INY	Progression de un de l'index Y
JMP	Saut à une nouvelle position
JSR	Saut à une nouvelle position, avec sauvegarde de l'adresse de retour

LDA	Chargement de la mémoire dans l'accumulateur
LDX	Chargement de la mémoire dans l'index X
LDY	Chargement de la mémoire dans l'index Y
LSR	Décalage d'un bit vers la droite (mémoire ou accumulateur)
NOP	Pas d'opération
ORA	Opération "OU" de la mémoire avec l'accumulateur
PHA	Accumulateur placé sur la pile
PHP	État de l'unité de traitement placé sur la pile
PLA	Accumulateur extrait de la pile
PLP	État de l'unité de traitement extrait de la pile
ROL	Rotation d'un bit vers la gauche (mémoire ou accumulateur)
ROR	Rotation d'un bit vers la droite (mémoire ou accumulateur)
RTI	Retour d'une interruption
RTS	Retour d'un sous-programme
SBC	Soustraction de la mémoire de l'accumulateur, avec retenue
SEC	Établissement de l'indicateur de report
SED	Établissement du mode décimal
SEI	Établissement de l'état d'invalidation d'interruption
STA	Stockage de l'accumulateur en mémoire
STX	Stockage de l'index X en mémoire
STY	Stockage de l'index Y en mémoire
TAX	Transfert de l'accumulateur à l'index X
TAY	Transfert de l'accumulateur à l'index Y
TSX	Transfert du pointeur de pile à l'index X
TXA	Transfert de l'index X à l'accumulateur
TXS	Transfert de l'index X au pointeur de pile
TYA	Transfert de l'index Y à l'accumulateur

MODÈLE DE PROGRAMMATION



JEU D'INSTRUCTIONS—CODES D'OPÉRATION, TEMPS

INSTRUCTIONS		Immédiat			Absolu			Page zéro			Accum.			Implicite			(Ind.)X			(Ind.)Y		
Mnémonique	Opération	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#
ADC	A + M + C→A (4) (1)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2
AND	A∧M→A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2
ASL	C←[7 0]←0				0E	6	3	06	5	2	0A	2	1									
BCC	BRANCHEMENT SUR C = 0(2)																					
BCS	BRANCHEMENT SUR C = 1(2)																					
BEO	BRANCHEMENT SUR Z = 1(2)																					
BIT	A∧M				2C	4	3	24	3	2												
BMI	BRANCHEMENT SUR N = 1(2)																					
BNE	BRANCHEMENT SUR Z = 0(2)																					
BPL	BRANCHEMENT SUR N = 0(2)																					
BRK	(Voir fig. 1)													00	7	1						
BVC	BRANCHEMENT SUR V = 0(2)																					
BVS	BRANCHEMENT SUR V = 1(2)																					
CLC	0→C													18	2	1						
CLD	0→D													D8	2	1						
CLI	0→I													58	2	1						
CLV	0→V													B8	2	1						
CMP	A − M (1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2
CPX	X − M	E0	2	2	EC	4	3	E4	3	2												
CPY	Y − M	C0	2	2	CC	4	3	C4	3	2												
DEC	M − 1→M				CE	6	3	C6	5	2												
DEX	X − 1→X													CA	2	1						
DEY	Y − 1→Y													B8	2	1						
EOR	A∀M→A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2
INC	M + 1→M				EE	6	3	E6	5	2												
INX	X + 1→X													E8	2	1						
INY	Y + 1→Y													C8	2	1						
JMP	SAUT NOUVELLE POS.				4C	3	3															
JSR	(Voir fig. 2) SAUT SOUS-PROG.				20	6	3															
LDA	M→A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2

Mnémonique	Z, page, X			Abs. X			Abs. Y			Relatif			Indirect			Z, page, Y			CODES DE CONDITION					
	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N	Z	C	I	D	V
ADC	75	4	2	7D	4	3	79	4	3										✓	✓	✓	—	—	✓
AND	35	4	2	3D	4	3	39	4	3										✓	✓	—	—	—	—
ASL	16	6	2	1E	7	3													✓	✓	✓	—	—	—
BCC										90	2	2							—	—	—	—	—	—
BCS										B0	2	2							—	—	—	—	—	—
BEO										F0	2	2							—	—	—	—	—	—
BIT																			M_7	✓	—	—	—	M_6
BMI										30	2	2							—	—	—	—	—	—
BNE										D0	2	2							—	—	—	—	—	—
BPL										10	2	2							—	—	—	—	—	—
BRK																			—	—	—	1	—	—
BVC										50	2	2							—	—	—	—	—	—
BVS										70	2	2							—	—	—	—	—	—
CLC																			—	—	0	—	—	—
CLD																			—	—	—	—	0	—
CLI																			—	—	—	0	—	—
CLV																			—	—	—	—	—	0
CMP	D5	4	2	DD	4	3	D9	4	3										✓	✓	✓	—	—	—
CPX																			✓	✓	✓	—	—	—
CPY																			✓	✓	✓	—	—	—
DEC	D6	6	2	DE	7	3													✓	✓	—	—	—	—
DEX																			✓	✓	—	—	—	—
DEY																			✓	✓	—	—	—	—
EOR	55	4	2	5D	4	3	59	4	3										✓	✓	—	—	—	—
INC	F6	6	2	FE	7	3													✓	✓	—	—	—	—
INX																			✓	✓	—	—	—	—
INY																			✓	✓	—	—	—	—
JMP													6C	5	3				—	—	—	—	—	—
JSR																			—	—	—	—	—	—
LDA	B5	4	2	BD	4	3	B9	4	3										✓	✓	—	—	—	—

D'EXÉCUTION, CONDITIONS DE MÉMOIRE

INSTRUCTIONS		Immédiat			Absolu			Page zéro			Accum.			Implicite			(Ind.)X			(Ind.)Y			Z. page, X			Abs. X			Abs. Y			Relatif			Indirect			Z. Page, Y			CODES DE CONDITION					
Mnémonique	Opération	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N	Z	C	I	D	V
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	✓	✓	—	—	—	—
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3													✓	✓	—	—	—	—
LSR	0 → [7 … 0] → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0	✓	✓	—	—	—
NOP	PAS D'OPÉRATION													EA	2	1																									—	—	—	—	—	—
ORA	A V M → A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										✓	✓	—	—	—	—
PHA	A → M_S S − 1 → S													48	3	1																									—	—	—	—	—	—
PHP	P → M_S S − 1 → S													08	3	1																									—	—	—	—	—	—
PLA	S + 1 → S M_S → A													68	4	1																									✓	✓	—	—	—	—
PLP	S + 1 → S M_S → P													28	4	1																									(RÉTABLI)					
ROL	← [7 … 0] ← [C] ←				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													✓	✓	✓	—	—	—
ROR	→ [C] → [7 … 0] →				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													✓	✓	✓	—	—	—
RTI	(Voir fig. 1) RETOUR ENT													40	6	1																									(RÉTABLI)					
RTS	(Voir fig. 2) RETOUR SOUS-PROG													60	6	1																									—	—	—	—	—	—
SBC	A − M − $\bar{C}$ → A (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										✓	✓	(3)	—	—	✓
SEC	1 → C													38	2	1																									—	—	1	—	—	—
SED	1 → D													F8	2	1																									—	—	—	—	1	—
SEI	1 → I													78	2	1																									—	—	—	1	—	—
STA	A → M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3										—	—	—	—	—	—
STX	X → M				8E	4	3	86	3	2																												96	4	2	—	—	—	—	—	—
STY	Y → M				8C	4	3	84	3	2													94	4	2																—	—	—	—	—	—
TAX	A → X													AA	2	1																									✓	✓	—	—	—	—
TAY	A → Y													A8	2	1																									✓	✓	—	—	—	—
TSX	S → X													BA	2	1																									✓	✓	—	—	—	—
TXA	X → A													8A	2	1																									✓	✓	—	—	—	—
TXS	X → S													9A	2	1																									—	—	—	—	—	—
TYA	Y → A													98	2	1																									✓	✓	—	—	—	—

(1) AJOUTER 1 À "N" SI ON A PASSÉ LA LIMITE DE PAGE.
(2) AJOUTER 1 À "N" SI LE BRANCHEMENT SE FAIT SUR LA MÊME PAGE.
AJOUTER 2 À "N" SI LE BRANCHEMENT SE FAIT SUR UNE AUTRE PAGE.
(3) NON-REPORT = RETENUE.
(4) EN MODE DÉCIMAL, L'INDICATEUR Z EST INVALIDE.
VÉRIFIER LE RÉSULTAT ZÉRO DANS L'ACCUMULATEUR.

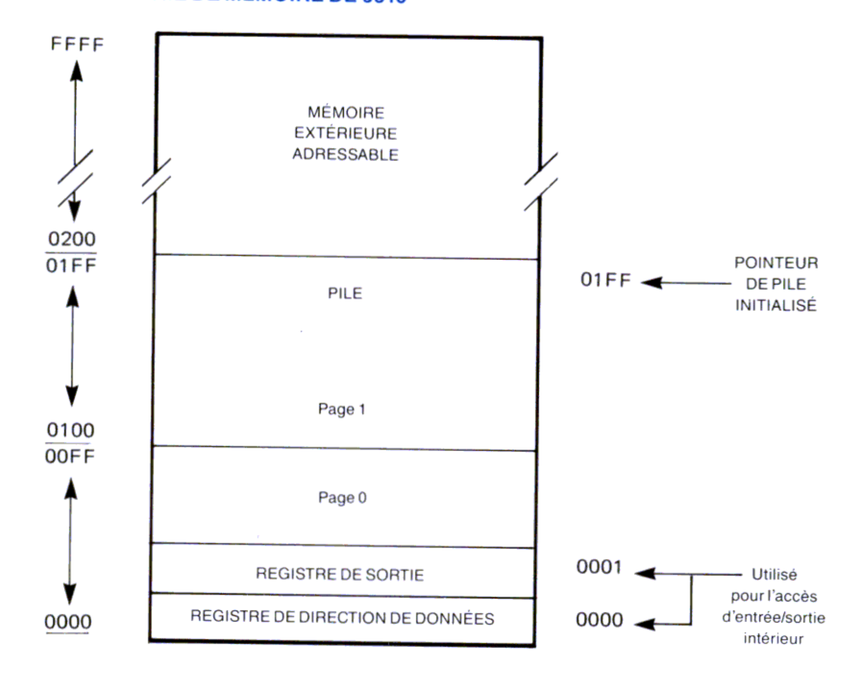
X INDEX X
Y INDEX Y
A ACCUMULATEUR
M MÉMOIRE D'APRÈS L'ADRESSE RÉELLE
M_S MÉMOIRE D'APRÈS LE POINTEUR DE PILE

\+ ADDITION
\- SOUSTRACTION
∧ ET
∨ OU
⊻ OU EXCLUSIF

✓ MODIFIÉ
— NON MODIFIÉ
M_7 BIT 7 DE MÉMOIRE
M_6 BIT 6 DE MÉMOIRE
N NOMBRE DE CYCLES
\# NOMBRE D'OCTETS

REMARQUE: LE GROUPE DES SEMI-CONDUCTEURS COMMODORE décline toute responsabilité relative à l'utilisation des codes d'opération non définis.

TOPOGRAPHIE DE MÉMOIRE DE 6510



REMARQUES SUR LES APPLICATIONS

Le positionnement du registre de sortie à l'accès d'entrée/sortie intérieur dans la page zéro rehausse les puissantes instructions d'adressage de page zéro du 6510.

En affectant des broches d'entrée/sortie comme entrées (avec le registre de direction de données), l'utilisateur peut changer le contenu de l'adresse 0001 (registre de sortie) à l'aide de dispositifs périphériques. La possibilité de changement de ces contenus par des entrées périphériques, avec les instructions d'adressage indirect de page zéro, offre des techniques de programmation nouvelles et flexibles inconnues jusqu'à présent.

LE GROUPE DES SEMI-CONDUCTEURS COMMODORE se réserve le droit d'apporter des changements à tout produit mentionné dans le présent guide pour en améliorer la fiabilité, le fonctionnement et la conception. **LE GROUPE DES SEMI-CONDUCTEURS COMMODORE** décline toute responsabilité découlant de l'application ou de l'utilisation de tout produit ou circuit décrit dans le présent guide; il n'accorde pas non plus d'autorisation dans le cadre de ses droits de brevet ni des droits de tierces personnes.

ANNEXE M

SPÉCIFICATIONS DE LA MICROPLAQUETTE D'ADAPTATEUR D'INTERFACE COMPLEXE (CIA) 6526

DESCRIPTION

L'adaptateur d'interface complexe (CIA) 6526 est un dispositif d'interface de périphérique 65XX compatible par bus doté de nombreuses possibilités d'entrée/sortie et de synchronisation.

CARACTÉRISTIQUES

- 16 lignes d'entrée/sortie programmables séparément
- Établissement de liaison 8 ou 16 bits à la lecture ou écriture
- 2 minuteries de 16 bits indépendantes pouvant être reliées
- Horloge à cycle 24 heures (AM/PM) avec alarme programmable
- Registre à décalage de 8 bits pour entrée/sortie série
- Capacité de charge 2TTL
- Lignes d'entrée/sortie compatibles CMOS
- Exploitation avec 1 ou 2 MHz

DISPOSITION DES BROCHES

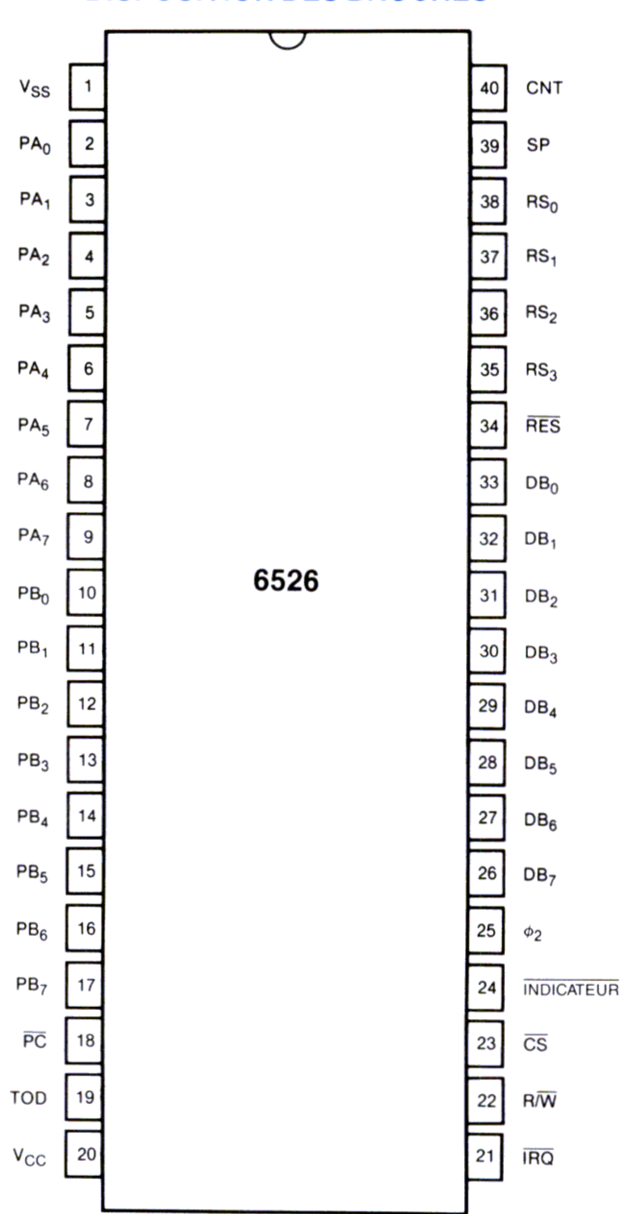
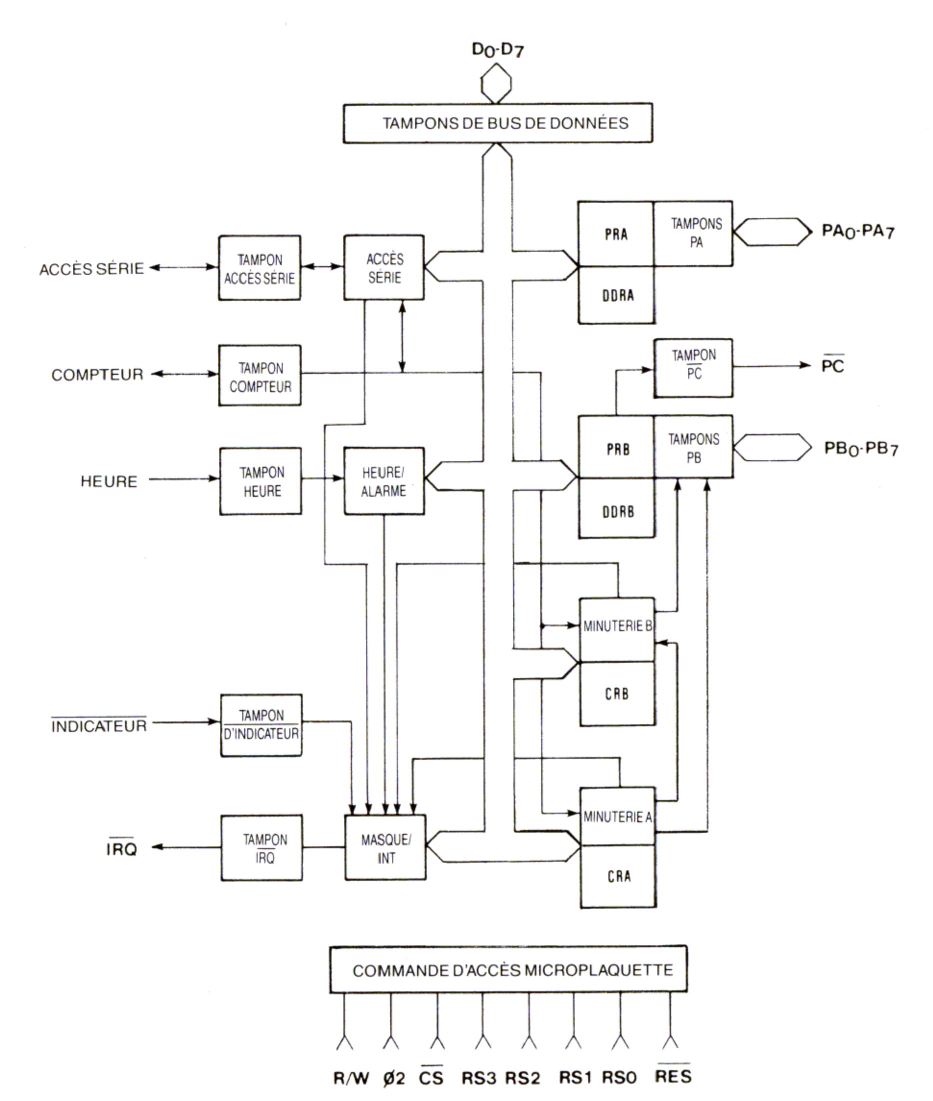


SCHÉMA DE PRINCIPLE
6526



VALEURS MAXIMALES

Tension d'alimentation, V_{CC}	−0.3 à +7.0 V
Tension d'entrée/sortie, V_{IN}	−0.3 à +7.0 V
Température d'utilisation, T_{OP}	0 à 70° C
Température de remisage, T_{STG}	−55 à 150° C

Toutes les entrées possèdent un circuit de protection évitant les dommages dus aux décharges statiques intenses. Éviter au maximum l'application inutile de tensions dépassant les limites permises.

REMARQUE

Les valeurs dépassant les limites maximales absolues peuvent causer des dommages définitifs au dispositif. Les valeurs indiquées sont des maximums. L'utilisation de ce dispositif avec ces valeurs ou toute autre valeur supérieure à celles indiquées dans les sections d'utilisation de ces spécifications n'est pas prévue; l'utilisation du dispositif aux valeurs maximales absolues pendant de longues périodes risque d'affecter défavorablement sa fiabilité.

CARACTÉRISTIQUES ÉLECTRIQUES
(V_{CC} ± 5%, V_{SS} = 0 V, T_A = 0—70°C)

CARACTÉRISTIQUE	SYMBOLE	MIN.	TYP.	MAX.	UNITÉ
Tension haute d'entrée	V_{IH}	+2.4	—	V_{CC}	V
Tension basse d'entrée	V_{IL}	−0.3	—	—	V
Courant de fuite d'entrée; $V_{IN} = V_{SS} + 5$ V (HEURE, R/W (lecture/écriture), $\overline{\text{INDICATEUR}}$, $\phi 2$, $\overline{\text{RES}}$, RS0-RS3, $\overline{\text{CS}}$)	I_{IN}	—	1.0	2.5	μA
Résistance chutrice d'entrée d'accès	R_{PI}	3.1	5.0	—	KΩ
Courant de fuite de sortie pour haute impédance (trois états); V_{IN} = 4 V à 2.4 V; (DB0-DB7, SP, CNT, $\overline{\text{IRQ}}$)	I_{TSI}	—	±1.0	±10.0	μA
Tension haute de sortie V_{CC} = MIN., $I_{CHARGE} <$ −200 μA (PA0-PA7, $\overline{\text{PC}}$, PB0-PB7, DB0-DB7)	V_{OH}	+2.4	—	V_{CC}	V
Tension basse de sortie V_{CC} = MIN., $I_{CHARGE} <$ 3.2 mA	V_{OL}	—	—	+0.40	V
Courant haut de sortie (source); $V_{OH} >$ 2.4 V (PA0-PA7, PB0-PB7, PC, DB0-DB7)	I_{OH}	−200	−1000	—	μA
Courant bas de sortie (débit); $V_{OL} <$ .4 V (PA0-PA7, $\overline{\text{PC}}$, PB0-PB7, DB0-DB7)	I_{OL}	3.2	—	—	mA
Capacité d'entrée	C_{In}	—	7	10	pf
Capacité de sortie	C_{OUT}	—	7	10	pf
Courant d'alimentation	I_{CC}	—	70	100	mA

DIAGRAMME DE SYNCHRONISATION D'ÉCRITURE DE 6526

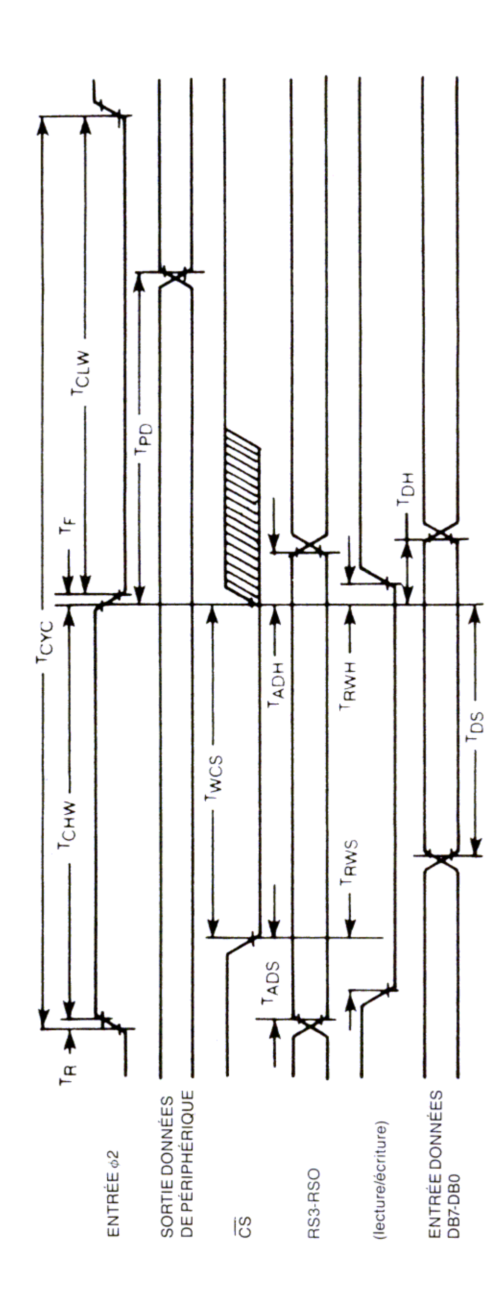
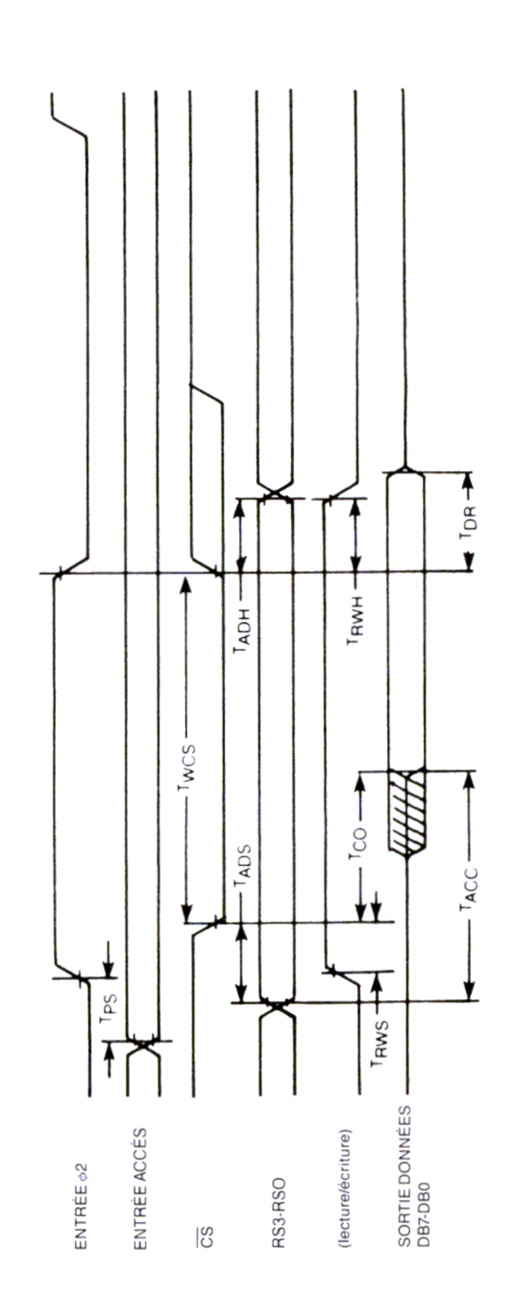


DIAGRAMME DE SYNCHRONISATION DE LECTURE DE 6526



SIGNAUX D'INTERFACE 6526

$\phi 2$—Entrée d'horloge

L'horloge $\phi 2$ est une entrée compatible TTL qui sert au fonctionnement des dispositifs internes et de référence de synchronisation pour les communications pour le bus de données du système.

$\overline{CS}$—Entrée de sélection de microplaquette

L'entrée $\overline{CS}$ commande l'activité de la microplaquette 6526. Avec un bas niveau sur $\overline{CS}$ quand $\phi 2$ est à l'état haut, le dispositif répond aux signaux sur les lignes de lecture/écriture et d'adresse (RS). Un état haut sur $\overline{CS}$ empêche ces lignes de commander la 6526. La ligne $\overline{CS}$ est normalement mise en fonction (état bas) à $\phi 2$ par la combinaison appropriée d'adresses.

R/W—Entrée de lecture/écriture

Le signal de lecture/écriture, qui vient normalement du microprocesseur, commande la direction des transferts de données de la 6526. Un état haut sur R/W correspond à une lecture (transfert de données sortant de la 6526); un état bas correspond à une écriture (transfert de données dans la 6526).

RS3-RS0—Entrées d'adresses

Les entrées d'adresses choisissent les registres internes, comme l'indique la topographie de registre.

DB7-DB0—Entrées/sorties de bus de données

Les huit broches de bus de données transfèrent l'information entre la 6526 et le bus de données du système. Ces broches sont des entrées à haute impédance, sauf si CS est à l'état bas et R/W et $\phi 2$ à l'état haut, pour lire le dispositif. Pendant cette lecture, les tampons de sortie du bus de données sont validés et envoient les données du registre choisi sur le bus de données du système.

$\overline{IRQ}$—Sortie de demande d'interruption

$\overline{IRQ}$ est une sortie débitrice ouverte, normalement raccordée à l'entrée d'interruption de l'unité de traitement. Une résistance chutrice externe maintient le signal à l'état haut permettant ainsi la réunion de plusieurs sorties $\overline{IRQ}$. La sortie $\overline{IRQ}$ est normalement hors fonction (haute impédance); elle est en fonction à l'état bas, comme l'indique la description du fonctionnement.

$\overline{RES}$—Entrée de remise à l'état initial

Un état bas à la broche de $\overline{RES}$ remet tous les registres internes à l'état initial. Les broches d'accès sont fixées, comme entrées et comme registres d'accès, à zéro, mais une lecture des accès ne retourne que des états hauts à cause de résistances chutrices passives. Les registres de commande de minuterie sont fixés à zéro et les bascules de minuterie sont toutes à un. Tous les autres registres sont remis à zéro.

CARACTÉRISTIQUES DE LA SYNCHRONISATION DE LA 6526

Symbole	Caractéristique	1 MHz		2 MHz		Unité
		MIN.	MAX.	MIN.	MAX.	
	Horloge $\phi 2$					
T_{CYC}	Durée de cycle	1000	20,000	500	20,000	ns
T_R, T_F	Durée de montée et de descente	—	25	—	25	ns
T_{CHW}	Largeur d'impulsion d'horloge (État haut)	420	10,000	200	10,000	ns
T_{CLW}	Largeur d'impulsion d'horloge (État bas)	420	10,000	200	10,000	ns
	Cycle d'écriture					
T_{PD}	Retard de sortie de $\phi 2$	—	1000	—	500	ns
T_{WCS}	$\overline{CS}$ à l'état bas pendant que $\phi 2$ est à l'état haut	420	—	200	—	ns
T_{ADS}	Temps d'établissement d'adresse	0	—	0	—	ns
T_{ADH}	Temps de maintien d'adresse	10	—	5	—	ns
T_{RWS}	Temps d'établissement lecture/écriture	0	—	0	—	ns
T_{RWH}	Temps de maintien lecture/écriture	0	—	0	—	ns
T_{DS}	Temps d'établissement de bus de données	150	—	75	—	ns
T_{DH}	Temps de maintien de bus de données	0	—	0	—	ns
	Cycle de lecture					
T_{PS}	Temps d'établissement d'accès	300	—	150	—	ns
$T_{WCS}(2)$	$\overline{CS}$ à l'état bas pendant que $\phi 2$ à l'état haut	420	—	20	—	ns
T_{ADS}	Temps d'établissement d'adresse	0	—	0	—	ns
T_{ADH}	Temps de maintien d'adresse	10	—	5	—	ns

Symbole	Caractéristique	1 MHz		2 MHz		Unité
		MIN.	MAX.	MIN.	MAX.	
T_{RWS}	Temps d'établissement lecture/écriture	0	—	0	—	ns
T_{RWH}	Temps de maintien lecture/écriture	0	—	0	—	ns
T_{ACC}	Accès de données de RS3-RS0	—	550	—	275	ns
T_{CO}(3)	Accès de données de CS	—	320	—	150	ns
T_{DR}	Temps de sortie des données	50	—	25	—	ns

REMARQUES: 1—Toutes les synchronisations s'entendent de V_{IL} max. et V_{IH} min. aux entrées et de V_{OL} max. et V_{OH} min. aux sorties.

2—T_{WCS} est mesurée de $\phi 2$ état haut ou $\overline{CS}$ état bas, suivant celui se produisant en dernier. $\overline{CS}$ doit être à l'état bas jusqu'à la fin au moins de $\phi 2$ à l'état haut.

3—T_{CO} est mesurée de $\phi 2$ état haut ou $\overline{CS}$ état bas, suivant celui se produisant en dernier. Les données valides ne sont disponibles qu'après T_{ACC} ou T_{CO}, suivant celle se produisant en dernier.

TOPOGRAPHIE DES REGISTRES

RS3	RS2	RS1	RS0	REG	NOM	
0	0	0	0	0	PRA	REGISTRE A DE DONNÉES DE PÉRIPHÉRIQUE
0	0	0	1	1	PRB	REGISTRE B DE DONNÉES DE PÉRIPHÉRIQUE
0	0	1	0	2	DDRA	REGISTRE A DE DIRECTION DE DONNÉES
0	0	1	1	3	DDRB	REGISTRE B DE DIRECTION DE DONNÉES
0	1	0	0	4	TA LO	REGISTRE BAS DE MINUTERIE A
0	1	0	1	5	TA HI	REGISTRE HAUT DE MINUTERIE A
0	1	1	0	6	TB LO	REGISTRE BAS DE MINUTERIE B
0	1	1	1	7	TB HI	REGISTRE HAUT DE MINUTERIE B
1	0	0	0	8	TOD 10THS	REGISTRE DES DIXIÈMES DE SECONDE
1	0	0	1	9	TOD SEC	REGISTRE DES SECONDES
1	0	1	0	A	TOD MIN	REGISTRE DES MINUTES
1	0	1	1	B	TOD HR	REGISTRE DES HEURES—AM/PM
1	1	0	0	C	SDR	REGISTRE DES DONNÉES SÉRIE
1	1	0	1	D	ICR	REGISTRE DE COMMANDE D'INTERRUPTION
1	1	1	0	E	CRA	REGISTRE DE COMMANDE A
1	1	1	1	F	CRB	REGISTRE DE COMMANDE B

DESCRIPTION DU FONCTIONNEMENT DE LA 6526

Accès d'entrée/sortie (PRA, PRB, DDRA, DDRB).

Les accès A et B se composent chacun d'un registre de données de périphérique (PR) de 8 bits et d'un registre de direction de données (DDR) de 8 bits. Si un bit du registre de direction de données est fixé à un, le bit correspondant du registre de données de périphérique donne une sortie; si un bit de registre de direction de données est fixé à zéro, le bit de registre de données de périphérique correspondant devient une entrée. Avec une instruction de lecture (READ), le registre de données de périphérique correspond aux informations présentes sur les broches d'accès réel (PA0-PA7,PB0-PB7) pour les bits d'entrée et de sortie. Les accès A et B ont des dispositifs de montée passifs et actifs permettant la compatibilité CMOS et TTL. Les deux accès permettent deux attaques de charge TTL. En dehors du fonctionnement d'entrée/sortie normal, PB6 et PB7 assurent aussi des fonctions de sortie de minuterie.

Établissement de liaison

L'établissement de liaison sur les transferts de données peut se faire à l'aide de la broche de sortie $\overline{\text{PC}}$ et de la broche d'entrée d'indicateur $(\overline{\text{FLAG}})$. $\overline{\text{PC}}$ passe à l'état bas pendant un cycle à la suite d'une lecture ou d'une écriture à l'accès B. Ce signal peut servir à indiquer des "données prêtes" à l'accès B ou des "données acceptées" de cet accès. L'établissement de liaison sur les transferts de données 16 bits (avec accès A et accès B) est permis par la lecture ou l'écriture de l'accès A en premier. L'indicateur $(\overline{\text{FLAG}})$ est une entrée sensible à front négatif qui peut servir de sortie PC pour une autre 6526 ou d'entrée d'interruption d'usage général. Une transition négative de l'indicateur $(\overline{\text{FLAG}})$ met le bit d'interruption d'indicateur à un.

REG.	NOM	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	PRA	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
1	PRB	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0
2	DDRA	DPA_7	DPA_6	DPA_5	DPA_4	DPA_3	DPA_2	DPA_1	DPA_0
3	DDRB	DPB_7	DPB_6	DPB_5	DPB_4	DPB_3	DPB_2	DPB_1	DPB_0

Minuteries d'intervalle (minuterie A et minuterie B)

Chaque minuterie d'intervalle se compose d'un compteur de minuterie à lecture seulement de 16 bits et d'une bascule de minuterie à écriture seulement de 16 bits. Les données écrites vers la minuterie sont verrouillées dans la bascule de minuterie alors que les données lues de la minuterie correspondent au contenu présent du compteur de minuterie. On peut utiliser indépendamment les minuteries ou les relier pour les opérations plus longues. Les différents modes de minuterie permettent la création de longs retards, d'impulsions de largeur variable, de trains d'impulsion et de formes d'onde de fréquence

variable. Avec l'entrée CNT, les minuteries peuvent compter des impulsions externes ou mesurer la fréquence, la largeur d'impulsion et les retards de signaux externes. Chaque minuterie possède un registre de commande connexe qui permet la commande indépendante des fonctions suivantes:

Départ/arrêt

Un bit de commande permet à tout moment la mise en marche ou l'arrêt de la minuterie par le microprocesseur.

Marche/arrêt PB:

Un bit de commande permet à la sortie de minuterie d'apparaître sur une ligne de sortie d'accès B (PB6 pour minuterie A et PB7 pour minuterie B). Cette fonction est prioritaire sur le bit de commande DDRB et force la ligne PB appropriée vers une sortie.

Bascule/impulsion

Un bit de commande choisit la sortie appliquée à l'accès B. À chaque dépassement de capacité négatif de minuterie, la sortie peut soit basculer, soit créer une seule impulsion positive d'une durée d'un cycle. La sortie de bascule est fixée à l'état haut quand la minuterie est mise en marche et est fixée à l'état bas par RES.

Pas à pas/continu

Un bit de commande choisit l'un ou l'autre des modes de minuterie. En mode pas à pas, la minuterie compte à rebours, de la valeur verrouillée jusqu'à zéro, crée une interruption, recharge la valeur verrouillée puis s'arrête. En mode continu, la minuterie compte de la valeur verrouillée à zéro, crée une interruption, recharge la valeur verrouillée et répète le processus en continu.

Chargement forcé

Un bit échantillon permet le chargement à tout moment de la bascule de minuterie dans le compteur de minuterie, avec la minuterie en ou hors fonction.

Mode d'entrée:

Les bits de commande permettent de choisir l'horloge utilisée pour diminuer la minuterie. La minuterie A peut compter les impulsions d'horloge $\phi 2$ ou les impulsions extérieures appliquées à la broche CNT. La minuterie B peut compter les impulsions $\phi 2$, les impulsions CNT extérieures, les impulsions de dépassement de capacité négatif de minuterie A ou de minuterie A quand la broche CNT est tenue à l'état haut.

La bascule de minuterie est chargée dans la minuterie pour tout dépassement de capacité négatif de minuterie, tout chargement forcé ou à la suite d'une écriture à l'octet de poids fort du précadreur quand la minuterie est à l'arrêt. Si la minuterie est en marche, une écriture à l'octet de poids fort charge la bascule de minuterie, mais ne recharge pas le compteur.

LECTURE (MINUTERIE)

REG.	NOM								
4	TA LO	TAL_7	TAL_6	TAL_5	TAL_4	TAL_3	TAL_2	TAL_1	TAL_0
5	TA HI	TAH_7	TAH_6	TAH_5	TAH_4	TAH_3	TAH_2	TAH_1	TAH_0
6	TB LO	TBL_7	TBL_6	TBL_5	TBL_4	TBL_3	TBL_2	TBL_1	TBL_0
7	TB HI	TBH_7	TBH_6	TBH_5	TBH_4	TBH_3	TBH_2	TBH_1	TBH_0

ÉCRITURE (PRÉCADREUR)

REG.	NOM								
4	TA LO	PAL_7	PAL_6	PAL_5	PAL_4	PAL_3	PAL_2	PAL_1	PAL_0
5	TA HI	PAH_7	PAH_6	PAH_5	PAH_4	PAH_3	PAH_2	PAH_1	PAH_0
6	TB LO	PBL_7	PBL_6	PBL_5	PBL_4	PBL_3	PBL_2	PBL_1	PBL_0
7	TB HI	PBH_7	PBH_6	PBH_5	PBH_4	PBH_3	PBH_2	PBH_1	PBH_0

Horloge de l'heure (TOD)

L'horloge de l'heure est une minuterie spéciale pour les applications en temps réel. Cette horloge à cycle de 24 heures (AM/PM) a une résolution de 1/10 de seconde. Elle est divisée en 4 registres: dixièmes de seconde, secondes, minutes et heures. L'indicateur AM/PM est dans le bit le plus significatif du registre des heures pour faciliter le contrôle des bits. Chaque registre se lit en format décimal codé binaire pour simplifier la conversion avant la commande des affiches, etc. L'horloge requiert une entrée de niveau TTL (programmable) de 60 ou 50 Hz sur la broche TOD pour marquer l'heure avec précision. En plus de l'indication de l'heure, il est prévu une alarme programmable pour créer une interruption à un moment désiré. Les registres d'alarme se trouvent aux mêmes adresses que les registres TOD correspondants. L'accès à l'alarme est commandé par un bit de registre de commande. L'alarme est de type à écriture seulement; toute lecture d'une adresses TOD indique l'heure, quel que soit l'état du bit d'accès d'alarme.

Il faut suivre une suite spécifique d'opérations pour le réglage et la lecture corrects de l'horloge TOD. Celle-ci s'arrête automatiquement en cas d'écriture dans le registre des heures. L'horloge ne se remet en marche qu'après une écriture au registre des dixièmes de seconde. L'horloge doit ainsi toujours se remettre en marche au moment désiré. Un report d'un étage au suivant pouvant se produire à tout moment dans une opération de lecture, une fonction de verrouillage est incluse pour maintenir constantes toutes les informations d'heure pendant une séquence de lecture. Les quatre registres d'horloge TOD se verrouillent sur une lecture des heures et restent verrouillés jusqu'après une lecture

des dixièmes de seconde. L'horloge TOD continue à compter quand les registres de sortie sont verrouillés. Si l'on ne doit lire qu'un registre, il n'y a pas de problème de report et on peut lire le registre "à la volée", pourvu que toute lecture des heures soit suivie d'une lecture des dixièmes de seconde pour invalider le verrouillage.

LECTURE

REG.	NOM								
8	TOD 10THS	0	0	0	0	T_8	T_4	T_2	T_1
9	TOD SEC	0	SH_4	SH_2	SH_1	SL_8	SL_4	SL_2	SL_1
A	TOD MIN	0	MH_4	MH_2	MH_1	ML_8	ML_4	ML_2	ML_1
B	TOD HR	PM	0	0	HH	HL_8	HL_4	HL_2	HL_1

ÉCRITURE

CRB_7 = 0 TOD

CRB_7 = 1 ALARME

(MÊME FORMAT QUE LA LECTURE)

Accès série (SDR)

L'accès série est un système tamponné à registre à décalage synchrone à 8 bits. Un bit de commande choisit le mode d'entrée ou de sortie. En mode d'entrée, les données sur la broche SP sont décalées dans le registre à décalage avec le flanc montant du signal appliqué à la broche CNT. Après 8 impulsions CNT, les données du registre à décalage sont vidées dans le registre de données série et il se crée une interruption. En mode de sortie, la minuterie A sert au générateur de régime de bauds. Les données sont sorties sur la broche SP à la moitié du régime de dépassement de capacité négatif de la minuterie A. Le régime maximal possible de bauds correspond à $\phi 2$ divisé par 4, mais le régime maximal utile de bauds est déterminé par la charge de ligne et la vitesse de réaction du récepteur aux données d'entrée. La transmission commence à la suite d'une écriture au registre de données série (à condition que la minuterie A soit en fonction et en mode continu). Le signal d'horloge dérivé de la minuterie A apparaît comme sortie à la broche CNT. Les données dans le registre de données série sont chargées dans le registre à décalage puis sorties vers la broche SP quand il se produit une impulsion CNT. Les données sorties deviennent valides au flanc descendant de l'impulsion CNT et le reste jusqu'au flanc descendant suivant.

Après 8 impulsions CNT, il se crée une interruption qui indique que l'on peut envoyer d'autres données. Si le registre de données série a été chargé de nouvelles informations avant cette interruption, les nouvelles données sont automatiquement chargées dans le registre à décalage et la transmission continue. Si le microprocesseur reste un octet en avance sur le registre à décalage, la transmission se fait en continu. S'il n'y a pas d'autres données à transmettre après la 8e impulsion CNT, la broche CNT revient à l'état haut et la broche SP reste au niveau du dernier bit de données transmis. Les données du registre

de données série sont sorties avec le bit le plus significatif en premier; les données d'entrée série doivent aussi apparaître sous ce format.

La caractéristique bidirectionnelle de l'accès série et de l'horloge CNT permet de raccorder de nombreux dispositifs 6526 à un bus de communication série commun sur lequel une microplaquette 6526 sert de dispositif principal, d'horloge à décalage et de données de source; les autres microplaquettes 6526 servent de dispositifs asservis. Les sorties CNT et SP fonctionnent en drain ouvert pour permettre le bus commun. Le protocole de la sélection "principal/asservi" peut se transmettre sur le bus série ou par des lignes de liaison spécialisées.

REG. NOM

C	SDR	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Commande d'interruption (ICR)

La 6526 comprend cinq sources d'interruptions: dépassement de capacité négatif de la minuterie A, dépassement de capacité négatif de la minuterie B, alarme d'horloge TOD, accès série plein/vide et indicateur. Un seul registre fournit les informations de filtrage et d'interruption. Le registre de commande des interruptions se compose d'un registre de masque à écriture seulement et d'un registre de données à lecture seulement. Toute interruption met le bit correspondant à un dans le registre de données. Une interruption validée par le registre de masque met le bit IR (bit le plus significatif) à un dans le registre de données et met la broche $\overline{IRQ}$ à l'état bas. Dans un système à plusieurs microplaquettes, on peut interroger le bit IR pour détecter la microplaquette qui a créé une demande d'interruption. Le registre de données d'interruption est effacé et la ligne $\overline{IRQ}$ revient à l'état haut après une lecture du registre de données. Chaque interruption met un bit d'interruption à un, quel que soit le masque, et on peut masquer sélectivement chaque bit d'interruption pour éviter de produire une interruption de l'unité de traitement; de ce fait, il est possible de mélanger des interruptions interrogées avec des interruptions vraies. Toutefois, l'interrogation du bit IR amène l'effacement du registre de données; par la suite, il revient à l'utilisateur de protéger les informations contenues dans le registre de données en présence d'interruptions interrogées.

Le registre de masque permet la commande pratique des bits de masque individuels. Quand on écrit vers le registre de masque, si le bit 7 (établissement/effacement) des données écrites est un ZÉRO, tout bit de masque écrit avec un un est effacé; les bits de masque écrits avec un zéro ne sont pas touchés. Si le bit 7 des données écrites est un UN, tout bit de masque écrit avec un un est établi et les bits de masque écrits avec un zéro ne sont pas touchés. Pour qu'un indicateur d'interruption mette le bit IR à un et crée une demande d'interruption, le bit de masque correspondant doit être à un.

LECTURE (DONNÉES ENTIÈRES)

REG. NOM

D	ICR	IR	0	0	FLG	SP	ALRM	TB	TA

ÉCRITURE (MASQUE ENTIER)

REG. NOM

D	ICR	S/C	X	X	FLG	SP	ALRM	TB	TA

REGISTRES DE COMMANDE

La microplaquette 6526 contient les registres de commande CRA et CRB. CRA est rattaché à la minuterie A et CRB à la minuterie B. Le format des registres est le suivant:

CRA:

Bit	Nom	Fonction
0	START	1=DÉPART MINUTERIE A, 0=ARRÊT MINUTERIE A. Ce bit est automatiquement remis à 0 en mode de dépassement de capacité négatif en mode pas à pas.
1	PBON	1=SORTIE DE MINUTERIE A apparaissant sur PB6, 0=fonctionnement normal PB6.
2	OUTMODE	1=BASCULE, 0=IMPULSION
3	RUNMODE	1=PAS À PAS, 0=CONTINU
4	LOAD	1=CHARGEMENT FORCÉ (entrée d'échantillon, sans stockage des données; le bit 4 prend toujours un zéro et l'écriture d'un zéro n'a aucun effet).
5	INMODE	1=LA MINUTERIE A compte les transitions CNT positives, 0=LA MINUTERIE A compte les impulsions $\phi 2$
6	SPMODE	1=SORTIE D'ACCÈS SÉRIE (sources CNT pour horloge à décalage), 0=ENTRÉE D'ACCÈS SÉRIE (horloge à décalage extérieure requise).
7	TODIN	1=Horloge 50 Hz requise sur broche TOD pour heure précise, 0=Horloge 60 Hz requise sur broche TOD pour heure précise.

CRB:

Bit	Nom	Fonction
		(Les bits CRB0-CRB4 sont identiques aux bits CRA0-CRA4 pour la minuterie B, sauf que le bit 1 commande la sortie de la minuterie B sur PB7.)
5,6	INMODE	Les bits CRB5 et CRB6 choisissent l'un des quatre modes d'entrée pour la minuterie B dans les conditions suivantes:

CRB6	CRB5	
0	0	La minuterie B compte les impulsions $\phi 2$.
0	1	La minuterie B compte les transitions CNT positives.
1	0	La minuterie B compte les impulsions de dépassement de capacité négatif de la minuterie A.
1	1	La minuterie B compte les impulsions de dépassement de capacité négatif de la minuterie A quand la broche CNT est à l'état haut.

Bit	Nom	Fonction
7	ALARM	1=L'écriture aux registres TOD fixe l'alarme, 0=l'écriture aux registres TOD fixe l'horloge TOD.

REG.	NOM	ENTRÉE TOD	MODE SP	MODE D'ENTRÉE	CHARGE	MODE D'EXÉCUTION	MODE DE SORTIE	PB MARCHE	DÉPART
E	CRA	0=60Hz 1=50Hz	0=ENTRÉE 1=SORTIE	0=$\phi 2$ 1=CNT	1=CHARGEMENT FORCE ÉCHANTILLON	0=CONTINU 1=PAS À PAS	0=IMPULSION 1=BASCULE	0=PB_6 ARRÊT 1=PB_6 MARCHE	0=ARRÊT 1=DÉPART

TA

REG.	NOM	ALARME	ENTRÉE	MODE	CHARGE	MODE D'EXÉCUTION	MODE DE SORTIE	PB MARCHE	DÉPART
F	CFB	0=TOD 1= ALARME	0 1 1 1	0=$\phi 2$ 1=CNT 0=TA 1=CNT•TA	1=CHARGEMENT FORCÉ ÉCHANTILLON	0=CONTINU 1=PAS À PAS	0=IMPULSION 1=BASCULE	0=PB_7 MARCHE 1=PB_7 ON	0=ARRÊT 1=DÉPART

TB

Tous les bits de registre non utilisés ne sont pas affectés par une écriture et sont forcés à zéro par une lecture.

SPÉCIFICATIONS DE LA MICROPLAQUETTE (VIC-11) 6566/6567

La 6566 et la 6567 sont des dispositifs contrôleurs vidéo couleur universels qui s'utilisent dans les terminaux vidéo d'ordinateur et les jeux vidéo. Les deux dispositifs contiennent 47 registres de commande accessibles par un bus de microprocesseur 8 bits standard (65XX); ils peuvent accéder à 16 K de mémoire pour l'affichage des informations. Nous décrivons ici les divers modes et options d'exploitation.

MODE D'AFFICHAGE DE CARACTÈRES

En mode d'affichage de caractères, la 6566 ou la 6567 extraient les pointeurs de caractère de la zone de matrice vidéo de la mémoire et les traduisent en adresses de position de point de caractère dans la zone de base de caractère de 2048 octets de la mémoire. La matrice vidéo se compose de 1000 positions consécutives en mémoire qui contiennent chacune un pointeur de caractère de 8 bits. Dans la mémoire, la position de la matrice vidéo est définie par VM13—VM10 dans le registre 24 ($18). Les valeurs représentent les 4 bits les plus significatifs de l'adresse de matrice vidéo. Les 10 bits de poids faible viennent d'un compteur interne (VC3—VC0) qui passe par les 1000 positions de caractère. On peut remarquer que la 6566 ou la 6567 donnent 14 sorties d'adresse; de ce fait, il peut falloir un matériel de système complémentaire pour le décodage complet de la mémoire du système.

ADRESSE DU SYSTÈME DE POINTEURS DE CARACTÈRE

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Le pointeur de caractère de 8 bits permet de disposer simultanément d'un maximum de 256 définitions différentes de caractère. Chaque caractère est représenté par une matrice de points de 8 x 8 stockée dans la base de caractère sous forme de 8 octets consécutifs. La position de la base de caractère est définie par CB13—CB11, également dans le registre 24 ($18), qui indiquent les 3 bits les plus significatifs de l'adresse de base de caractère. Les 11 adresses de poids faible sont constituées par le pointeur de caractère de 8 bits de la matrice vidéo (D7—D0), qui choisit un caractère particulier, et par un compteur de trame de 3 bits (RC2—RC0), qui choisit l'un des 8 octets de caractère. Les caractères résultants sont mis en forme sur 25 rangées de 40 caractères chacune. En plus du pointeur de caractère de 8 bits, un demi-octet couleur de 4 bits est rattaché à chaque position de matrice vidéo (la mémoire de matrice vidéo doit avoir 12 bits de large) qui définit l'une des 16 couleurs pour chaque caractère.

ADRESSE DE DONNÉES DE CARACTÈRE

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	CB12	CB11	D7	D6	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

MODE DE CARACTÈRE STANDARD (MCM = BMM = ECM = 0)

En mode de caractère standard, les 8 octets consécutifs de la base de caractère sont affichés directement sur les 8 lignes dans chaque région de caractère. Un bit "0" amène l'affichage de la couleur d'arrière-plan n° 0 du registre 33 ($21); la couleur choisie par le demi-octet couleur (premier plan) est affichée avec un bit "1" (voir Table des codes de couleur).

FONCTION	BIT DE CARACTÈRE	COULEUR AFFICHÉE
Arrière-plan	0	Couleur d'arrière plan n° 0 (registre 33 ($21))
Premier plan	1	Couleur choisie par le demi-octet couleur de 4 bits

Chaque caractère a donc une couleur particulière déterminée par le demi-octet couleur (1 de 16) de 4 bits; tous les caractères partagent la couleur commune d'arrière-plan.

MODE DE CARACTÈRE MULTICOLORE (MCM = 1, BMM = ECM = 0)

Le mode multicolore apporte une plus grande flexibilité de couleurs. Il permet jusqu'à 4 couleurs dans chaque caractère, mais avec une définition réduite. On choisit le mode multicolore en fixant le bit MCM du registre 22 ($16) à "1" pour interpréter de façon différente les données de points stockées dans la base de caractère. Si le bit le plus significatif du demi-octet couleur est "0", le caractère est affiché tel qu'indiqué en mode de caractère standard; il est possible de mélanger les deux modes (les huit couleurs de poids faible sont toutefois les seules disponibles). Si le bit le plus significatif du demi-octet couleur est "1" (si MCM:bit le plus significatif (CM) = 1), les bits de caractère sont interprétés en mode multicolore de la façon suivante:

FONCTION	PAIRE DE BITS DE CARACTÈRE	COULEUR AFFICHÉE
Arrière-plan	00	Couleur d'arrière-plan n° 0 (registre 33 ($21))
Arrière-plan	01	Couleur d'arrière-plan n° 1 (registre 34) ($22))
Premier plan	10	Couleur d'arrière-plan n° 2 (registre 35) ($23))
Premier plan	11	Couleur spécifiée par les trois bits les moins significatifs du demi-octet couleur

Deux bits étant requis pour spécifier une couleur de point, le caractère est maintenant affiché sous forme d'une matrice de 4 x 8, chaque point ayant deux fois la taille horizontale du mode standard. Il faut toutefois remarquer que chaque région de caractère peut maintenant contenir 4 couleurs différentes, deux pour le premier plan et deux pour l'arrière-plan (voir priorité des blocs-objets mobiles—MOB).

MODE DE COULEUR ÉTENDU (ECM = 1, BMM = MCM = 0)

Le mode de couleur étendu permet un choix de couleurs séparées d'arrière-plan pour chaque région de caractère avec la définition de caractère normale de 8 x 8. On choisit ce mode en fixant le bit ECM de registre 17 ($11) à "1". Les données de point de caractère

sont affichées comme en mode standard (la couleur de premier plan déterminée par le demi-octet couleur est affichée avec un bit de données "1"), mais les deux bits les plus significatifs du pointeur de caractère servent à choisir la couleur d'arrière-plan de chaque région de caractère dans les conditions suivantes:

PAIRE DE BITS LES PLUS SIGNIFICATIFS DE POINTEUR DE CARACTÈRE	COULEUR D'ARRIÈRE-PLAN AFFICHÉE POUR BIT 0
00	Couleur d'arrière-plan nº 0 (registre 33 ($21))
01	Couleur d'arrière-plan nº 1 (registre 34 ($22))
10	Couleur d'arrière-plan nº 2 (registre 35 ($23))
11	Couleur d'arrière-plan nº 3 (registre 36 ($24))

Les deux bits les plus significatifs des pointeurs de caractère servent aux informations de couleur; on ne dispose donc que de 64 définitions différentes de caractère. La 6566 ou la 6567 forcent CB10 et CB9 à l'état "0", quelles que soient les valeurs originales de pointeur, de façon à pouvoir accéder uniquement aux 64 premières définitions de caractère. En mode de couleur étendu, chaque caractère possède une des 16 couleurs de premier plan définies séparément et une des 4 couleurs disponibles d'arrière-plan.

REMARQUE: On ne doit pas valider simultanément les modes multicolore et de couleur étendu.

MODE DE TOPOGRAPHIE BINAIRE

En mode de topographie binaire, la 6566 ou la 6567 extraient les données de la mémoire de façon différente, de façon qu'il existe une correspondance exacte entre chaque point affiché et un bit de mémoire. Le mode de topographie binaire donne une définition de 320 points horizontaux par 200 points verticaux, contrôlés séparément. On choisit le mode de topographie binaire en fixant le bit BMM dans le registre 17 ($11) à "1". On accède encore à la matrice vidéo comme en mode de caractères, mais ses données ne sont plus interprétées comme des pointeurs de caractère, mais plutôt comme des données couleur. Le compteur de matrice vidéo sert aussi d'adresse pour extraire les données de points pour l'affichage à partir de la base d'affichage de 8000 octets. L'adresse de base d'affichage est constituée de la façon suivante:

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

VCx correspond aux sorties de compteur de matrice vidéo, RCx au compteur de ligne de trame à 3 bits; CB13 vient du registre 24 ($18). Le compteur de matrice vidéo passe par les mêmes 40 positions pour huit lignes de trame et continue aux 40 positions suivantes à chaque huitième ligne; le compteur de trame augmente à chaque ligne vidéo horizontale (ligne de trame). Cet adressage se traduit par la mise en forme de chaque série de huit positions de mémoire séquentielle en bloc de données de 8 x 8 sur l'écran vidéo.

MODE DE TOPOGRAPHIE BINAIRE STANDARD (BMM = 1, MCM = 0)

Quand on utilise le mode de topographie binaire standard, les informations de couleur ne viennent que des données stockées dans la matrice vidéo (le demi-octet couleur est ignoré). Les huit bits sont répartis en deux demi-octets de 4 bits qui permettent le choix indépendant de deux couleurs dans chaque bloc de 8 x 8 points. Quand un bit de la mémoire d'affichage est à "0", la couleur du point de sortie est fixée par le demi-octet le moins significatif (poids faible). De même un bit de mémoire d'affichage "1" choisit la couleur de sortie déterminée par le demi-octet le plus significatif (poids fort).

BIT	COULEUR D'AFFICHAGE
0	Demi-octet de poids faible du pointeur de matrice vidéo
1	Demi-octet de poids fort du pointeur de matrice vidéo

MODE DE TOPOGRAPHIE BINAIRE MULTICOLORE (BMM = MCM = 1)

On choisit le mode de topographie binaire multicolore en fixant le bit MCM dans le registre 22 ($16) à "1", en même temps que le bit BMM. Le mode multicolore emploie les mêmes séquences d'accès de mémoire que le mode de topographie binaire standard, mais il interprète les données de points de la façon suivante:

PAIRE DE BITS	COULEUR D'AFFICHAGE
00	Couleur d'arrière-plan nº 0 (registre 33 ($21))
01	Demi-octet de poids fort du pointeur de matrice vidéo
10	Demi-octet de poids faible du pointeur de matrice vidéo
11	Demi-octet couleur de matrice vidéo

Il faut noter que le demi-octet couleur (DB11—DB8) sert au mode de topographie binaire multicolore. Cette fois encore, avec deux bits servant à choisir une couleur de point, la taille de point horizontal est doublée et se traduit par une définition d'écran de 160 points

horizontaux par 200 points verticaux. Avec le mode de topographie binaire multicolore, on peut afficher trois couleurs choisies séparément dans chaque bloc de 8 x 8, en plus de la couleur d'arrière-plan.

BLOCS-OBJETS MOBILES

Le bloc-objet mobile (MOB) est un type spécial de caractère que l'on peut afficher en tout point de l'écran sans les limitations inhérentes aux modes de caractère et de topographie binaire. On peut afficher simultanément jusqu'à 8 blocs-objets mobiles, chacun étant défini en mémoire par 63 octets qui sont affichés sous forme de tableau de 24 x 21 points (voir ci-dessous). Plusieurs caractéristiques spéciales rendent les blocs-objets mobiles particulièrement pratiques pour les graphiques et les jeux vidéo.

AFFICHAGE DE BLOCS-OBJETS MOBILES

OCTET	OCTET	OCTET
00	01	02
03	04	05
.	.	.
.	.	.
.	.	.
57	58	59
60	61	62

VALIDATION

On peut valider sélectivement chaque bloc-objet mobile pour l'affichage en fixant le bit de validation correspondant (MnE) à "1" dans le registre 21 ($15). Si le bit MnE est à "0", il ne se produit aucune opération intéressant le bloc-objet mobile à valider.

POSITION

Chaque bloc-objet mobile est disposé suivant son registre de positions X et Y (voir topographie des registres) avec une définition de 512 positions horizontales et 256 positions verticales. La position d'un bloc-objet mobile est déterminée par le point supérieur gauche du tableau. Les positions X 23 à 347 ($17 — $157) et les positions Y 50 à 249 ($32 — $F9) sont visibles. Les positions de bloc-objet mobile disponibles n'étant pas toutes entièrement visibles sur l'écran, on peut faire disparaître les blocs-objets mobiles de l'écran et les y faire apparaître en douceur.

COULEUR

Chaque bloc-objet mobile possède un registre séparé de 4 bits qui détermine sa couleur. Les deux modes de couleur de bloc-objet mobile sont:

BLOC-OBJET MOBILE STANDARD (MnMC = 0)

En mode standard, un bit "0" de données de bloc-objet mobile permet de voir les données d'arrière-plan par transparence. Un bit "1" affiche la couleur de bloc-objet mobile déterminée par le registre de couleur de bloc-objet mobile correspondant.

BLOC-OBJET MOBILE MULTICOLORE (MnMC = 1)

Chaque bloc-objet mobile peut être rendu séparément multicolore par les bits MnMC du registre multicolore 28 ($1C) de bloc-objet mobile. Si le bit MnMC est "1", le bloc-objet mobile correspondant est affiché en mode multicolore. Dans ce mode, les données de bloc-objet mobile sont interprétées par paires (comme les autres modes multicolores) de la façon suivante:

PAIRE DE BITS	COULEUR AFFICHÉE
00	Transparence
01	Bloc-objet mobile multicolore N° 0 (registre 37 ($25))
10	Bloc-objet mobile couleur (registres 39 à 46 ($27—$2E))
11	Bloc-objet mobile multicolore (registre 38 ($26))

Deux bits de données étant nécessaires à chaque couleur, la définition du bloc-objet mobile est ramenée à 12 x 21, chaque point horizontal étant porté à deux fois la dimension normale pour que la taille globale du bloc-objet mobile ne change pas. Il faut noter que l'on peut afficher jusqu'à trois couleurs dans chaque bloc-objet mobile (en plus de la transparence), mais deux des couleurs sont partagées par tous les blocs-objets mobiles multicolores.

GROSSISSEMENT

On peut sélectivement agrandir (x 2) chaque bloc-objet mobile dans les sens horizontal et vertical. Deux registres contiennent les commandes (MnXE, MnYE) pour la commande de grossissement:

REGISTRE	FONCTION
23 ($17)	Agrandissement horizontal MnXE—"1"=agrandissement; "0"=normal
29 ($1D)	Agrandissement vertical MnYE—"1"=agrandissement; "0"=normal

Quand on agrandit des blocs-objets mobiles, on n'obtient aucune amélioration de la définition. Le même tableau de 24 x 21 (12 x 21 en mode multicolore) est affiché, mais la taille globale des blocs-objets mobiles est doublée dans le sens désiré. Le point le plus petit de bloc-objet mobile peut s'agrandir jusqu'à quatre fois si le bloc-objet mobile est à la fois en mode multicolore et agrandi.

PRIORITÉ

On peut commander séparément la priorité de chaque bloc-objet mobile par rapport aux autres informations affichées en modes de caractères et de topographie binaire. La priorité de chaque bloc-objet mobile est fixée par le bit correspondant (MnDP) du registre 27 ($1B) de la façon suivante:

BIT DE REGISTRE	PRIORITÉ AUX DONNÉES DE CARACTÈRE OU DE TOPOGRAPHIE BINAIRE
0	Les données de bloc-objet mobile non transparent sont affichées (bloc-objet mobile à l'avant)
1	Les données de bloc-objet mobile non transparent sont affichées uniquement à la place de la couleur de fond n° 0 de la paire de bits multicolores 01 (bloc-objet mobile à l'arrière).

BLOC-OBJET MOBILE — PRIORITÉ DES DONNÉES D'AFFICHAGE

MnDP = 1	MnDP = 0
Bloc-objet mobile n	Premier plan
Premier plan	Bloc-objet mobile n
Arrière-plan	Arrière-plan

Les bits "0" ("00" en mode multicolore) de données de bloc-objet mobile sont transparents et permettent toujours l'affichage d'autres informations.

Les blocs-objets mobiles ont une priorité fixe les uns par rapport aux autres. Le bloc-objet mobile 0 a la priorité la plus élevée, le bloc-objet mobile 7 la plus basse. Quand les données (excepté les données de transparence) de deux blocs-objets mobiles coïncident, les données du bloc-objet mobile le plus bas sont affichées. En mode de caractères ou de topographie binaire, les priorités entre blocs-objets mobiles sont établies avant la résolution de priorité.

DÉTECTION DE COLLISION

Deux types de collision (ou coïncidence) de blocs-objets mobiles sont détectés: collision entre blocs-objets mobiles et collision de bloc-objet mobile avec les données d'affichage.

1) Il se produit une collision entre deux blocs-objets mobiles quand les données de sortie non transparentes de deux blocs-objets mobiles coïncident. La coïncidence des zones transparentes de blocs-objets mobiles ne crée pas une collision. En cas de collision, les bits (MnM) de bloc-objet mobile dans le registre 30 ($1E) de collision entre blocs-objets mobiles sont fixés à "1" pour les deux blocs-objets en collision. Quand il se produit une collision entre deux ou plusieurs blocs-objets mobiles, le bit de collision de bloc-objet est fixé à 1 pour chacun des blocs-objets. Les bits de collision restent à 1 jusqu'à la lecture du registre de collision; les bits sont ensuite automatiquement effacés. Les collisions entre blocs-objets mobiles sont détectées même si elles se produisent en dehors de l'écran.

2) L'autre cas correspond à la collision d'un bloc-objet mobile et des données d'affichage de premier plan des modes de caractère ou de topographie binaire. Le registre 31 ($1F) de collision bloc-objet mobile/données possède un bit (MnD) pour chaque bloc-objet qui est fixé à "1" quand le bloc-objet mobile et les données d'affichage n'appartenant pas à l'arrière-plan coïncident. Ici encore, la coïncidence des données transparentes n'amène pas de collision. Dans les applications spéciales, les données d'affichage de la paire de bits multicolores 0-1 ne causent pas non plus de collision. Cette caractéristique permet leur utilisation comme données d'affichage d'arrière- plan sans gêner les collisions véritables entre blocs-objets mobiles. Il peut se produire une collision entre bloc-objet mobile et données en dehors de l'écran, dans le sens horizontal, si les données d'affichage réelles sont sorties à une position hors de l'écran (voir défilement). Le registre de collision bloc-objet mobile/données s'efface aussi automatiquement à la lecture.

Les bascules d'interruption de collision sont à "1" quand le premier bit de l'un ou l'autre registre est fixé à "1". Quand un bit de collision dans un registre est à l'état haut, les collisions suivantes ne mettent pas la bascule d'interruption à "1" tant que ce registre de collision n'a pas été effacé à "0" par une lecture.

ACCÈS À LA MÉMOIRE DE BLOC-OBJET MOBILE

Les données de chaque bloc-objet mobile sont stockées dans 63 octets consécutifs de mémoire. Chaque bloc de données de bloc-objet mobile est défini par un pointeur de bloc-objet mobile, situé à la fin de la matrice vidéo. Il n'est utilisé que 1000 octets de la matrice vidéo dans les modes d'affichage normaux; on peut donc utiliser les positions de matrice vidéo 1016 à 1023 (base VM + $3F8 à base VM + $3FF) respectivement pour les pointeurs 0 à 7 de bloc-objet mobile. Le pointeur de bloc-objet mobile à huit bits de la matrice vidéo et les six bits du compteur d'octet bloc-objet mobile (pour adresser 63 octets) définissent la zone complète d'adresse de 14 bits.

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
MP7	MP6	MP5	MP4	MP3	MP2	MP1	MP0	MC5	MC4	MC3	MC2	MC1	MC0

Dans cette zone, MPx correspond aux bits de pointeur de bloc-objet mobile programmable de la matrice vidéo et MCx au bit de compteur de bloc-objet mobile de création interne. Les pointeurs de bloc-objet mobile sont lus de la matrice vidéo à la fin de chaque ligne de trame. Quand le registre de position Y d'un bloc-objet mobile correspond au compte de ligne de trame courant, la recherche réelle des données de bloc-objet commence. Les compteurs internes passent automatiquement par les 63 octets des données de blocs-objets mobiles et affichent trois octets sur chaque ligne de trame.

AUTRES CARACTÉRISTIQUES

EFFACEMENT DE L'ÉCRAN

On peut effacer un écran de visualisation en fixant le bit DEN dans le registre 17 ($11) à "0". Quand l'écran est effacé, il se couvre totalement de la couleur extérieure fixée dans le registre 32 ($20). Quand l'effacement est en cours, seul sont requis les accès de mémoire transparente (phase 1), permettant ainsi l'utilisation intégrale du bus du système par l'unité de traitement. Les données de bloc-objet mobile sont toutefois sollicitées si les blocs-objets mobiles ne sont pas également invalidés. Le bit DEN doit être fixé à "1" pour l'affichage vidéo normal.

SÉLECTION DE RANGÉES/COLONNES

L'affichage normal se compose de 25 rangées de 40 caractères (ou régions de caractères). Pour les conditions spéciales, on peut réduire la fenêtre d'affichage à 24 rangées de 38 caractères. Il n'y a aucun changement dans le format des informations affichées mais les caractères (bits) voisins de la zone de cadre extérieur sont maintenant couverts par le cadre. Les bits de sélection fonctionnent ainsi:

RSEL	NOMBRE DE RANGÉES	CSEL	NOMBRE DE COLONNES
0	24	0	38
1	25	1	40

Le bit RSEL est dans le registre 17 ($11) et le bit CSEL dans le registre 22 ($16). Pour l'affichage standard, on utilise normalement la plus grande fenêtre d'affichage. La fenêtre d'affichage plus petite s'emploie normalement avec le défilement.

DÉFILEMENT

On peut faire défiler les données d'affichage d'un espace de caractère complet dans les sens horizontal et vertical. Utilisé avec la fenêtre d'affichage plus petite (voir ci-dessus), le défilement permet de créer un déplacement uniforme des données d'affichage tout en mettant à jour la mémoire du système, uniquement si une nouvelle rangée (ou colonne) de caractères est requise. Le défilement sert aussi à centrer un affichage fixe dans la fenêtre.

BITS	REGISTRE	FONCTION
X2,X1,X0	22 ($16)	Position horizontale
Y2,Y1,Y0	17 ($11)	Position verticale

CRAYON LUMINEUX

L'entrée de crayon lumineux verrouille la position d'écran présente dans une paire de registres (LPX,LPY) en cas d'impulsion à front descendant. Le registre 19 ($13) de position X contrôle les huit bits les plus significatifs de la position X au moment de la transition. La position X étant définie par un compteur de 512 états (9 bits), on dispose d'une définition de 2 points horizontaux. De même, la position Y est verrouillée au registre 20 ($14), mais on obtient ici une définition de trame simple avec huit bits dans l'affichage visible. La bascule de crayon lumineux ne peut se déclencher qu'une fois par trame; les

déclenchements suivants dans la même trame n'ont aucun effet. De ce fait, on doit prendre plusieurs échantillons avant de mettre le crayon lumineux sur l'écran (en moyenne trois échantillons ou plus), suivant les caractéristiques du crayon utilisé.

REGISTRE DE TRAME

Le registre de trame comprend deux fonctions: une lecture du registre de trame 18 ($12) retourne les huit bits de poids faible de la position de trame courante (le bit le plus significatif RC8 se trouve dans le registre 17 ($11)). On peut interroger le registre de trame pour mettre les changements d'affichage en oeuvre en dehors de la zone visible afin d'éviter le scintillement. La fenêtre d'affichage visible va des trames 51 à 251 ($033—$0FB). Une écriture aux bits de trame (RC8 compris) est verrouillée pour l'utilisation dans une comparaison de trame interne. Quand la trame courante correspond à la valeur écrite, la bascule d'interruption de trame est fixée à "1".

REGISTRE D'INTERRUPTION

Le registre d'interruption indique l'état des quatre sources d'interruption. Dans le registre 25 ($19), une bascule d'interruption est fixée à "1" quand une source d'interruption a créé une demande d'interruption. Les quatre sources d'interruption sont:

BIT DE BASCULE	BIT DE VALIDATION	MISE À L'ÉTAT 1
IRST	ERST	Quand "compte de trame" = (compte de trame stocké)
IMDC	EMDC	Par le registre de collision bloc-objet mobile/données (première collision seulement)
IMMC	EMMC	Par le registre de collision entre blocs-objets mobiles (première collision seulement)
ILP	ELP	Par la première transition négative de l'entrée LP (une fois par trame)
IRQ		À l'état O par le jeu de bascule et validé (inversion de sortie/IRQ)

Pour valider une demande d'interruption et fixer la sortie IRQ à "0", le bit de validation d'interruption correspondant dans le registre 26 ($1A) doit être fixé à "1". Quand une bascule d'interruption est fixée à "1", on ne peut l'effacer qu'en écrivant "1" dans la bascule désirée du registre d'interruption. Cette caractéristique permet la manipulation sélective des interruptions vidéo sans logiciel, devant mémoriser les interruptions rapides.

RÉGÉNÉRATION DES MÉMOIRES RAM DYNAMIQUES

Un contrôleur de régénération de mémoires RAM dynamiques est intégré dans les dispositifs 6566 et 6567. Cinq adresses de rangées de 8 bits sont régénérées pour chaque ligne de trame. Ce régime garantit un retard maximal de 2.02 ms entre la régénération de toute adresse de rangée simple dans une disposition de régénération de 128 adresses (le retard maximal est de 3.66 ms dans une disposition de régénération de 256 adresses). Cette régénération est totalement transparente pour le système, car elle se produit pendant la phase 1 de l'horloge du système. Le dispositif 6567 crée les adresses RAS/ et CAS/ qui sont normalement reliées aux mémoires RAM dynamiques. RAS/ et CAS/ sont créées pour chaque phase 2 et chaque accès de données vidéo (régénération comprise) de manière que la régénération extérieure d'horloge ne soit pas requise.

REMISE À L'ÉTAT INITIAL

THÉORIE DU FONCTIONNEMENT

INTERFACE DE SYSTÈME

Les dispositifs contrôleurs vidéo 6566 et 6567 assurent une interaction spéciale avec le bus de données du système. Dans un système 65XX, les bus ne sont requis que pendant la phase 2 (horloge à l'état haut) du cycle. Les dispositifs 6566 et 6567 tirent parti de cette caractéristique en accédant normalement à la mémoire du système pendant la phase 1 (horloge à l'état bas) du cycle d'horloge. De ce fait, les opérations comme les extractions de données de caractères et la régénération de mémoire sont transparentes pour l'unité centrale et ne réduisent pas le débit de celle-ci. Les microplaquettes vidéo donnent les signaux de commande d'interface requis pour maintenir ce partage des bus.

Les dispositifs vidéo donnent le signal de commande de validation d'adresse qui sert à invalider les commandes de bus d'adresse d'unité de traitement et permet aux dispositifs vidéo d'accéder au bus d'adresse. La commande de validation d'adresse est active à l'état bas et permet la connexion directe à l'entrée de commande de validation d'adresse

de la famille 65XX. Le signal de commande de validation d'adresse est normalement en fonction pendant la phase 1 pour ne pas affecter le fonctionnement de l'unité de traitement. Grâce à ce "partage" des bus, tous les accès de mémoire doivent être exécutés en un demi-cycle. Les microplaquettes donnant une horloge de 1 MHz (qui doit être utilisé pour la phase 2 du système), un cycle de mémoire mesure 500 ns y compris l'établissement de l'adresse, l'accès aux données et l'établissement des données au dispositif de lecture.

Certaines opérations de la 6566 et de la 6567 doivent recevoir des données à un régime plus rapide qu'il n'est permis par la lecture uniquement pendant la durée de la phase 1; mentionnons en particulier l'accès des pointeurs de caractère de la matrice vidéo et l'extraction des données de bloc-objet mobile. De ce fait, on doit invalider l'unité de traitement et accéder aux données pendant le cycle d'horloge de phase 2 par l'intermédiaire du signal de bus disponible. La ligne de bus disponible, normalement à l'état haut, est amenée à l'état bas pendant la phase 1 pour indiquer que la microplaquette vidéo demande un accès de données de phase 2. Trois temps de phase 2 sont permis après l'état bas de ligne de bus disponible pour que l'unité de traitement termine les accès en cours à la mémoire. Au quatrième temps de phase 2, après l'état bas de la ligne de bus disponible, le signal de commande de validation d'adresse reste à l'état bas pendant la phase 2 pendant que la microplaquette vidéo extrait les données. La ligne de bus disponible est normalement reliée à l'entrée RDY d'une unité de traitement 65XX. Les extractions de pointeurs de caractère se font à chaque huitième ligne de trame, pendant la fenêtre d'affichage; il faut 40 accès consécutifs de phase 2 pour extraire les pointeurs de matrice vidéo. Les extractions des données de bloc-objet mobile nécessitent quatre accès de mémoire dans les conditions suivantes:

PHASE	DONNÉES	CONDITIONS
1	Pointeur de bloc-objet mobile	À chaque trame
2	Octet 1 de bloc-objet mobile	À chaque trame, pendant l'affichage du bloc-objet mobile
1	Octet 2 de bloc-objet mobile	À chaque trame, pendant l'affichage du bloc-objet mobile
2	Octet 3 de bloc-objet mobile	À chaque trame, pendant l'affichage du bloc-objet mobile

Les pointeurs de bloc-objet mobile sont extraits pendant une phase 1 sur deux, à la fin de chaque ligne de trame. Suivant le cas, on utilise les cycles complémentaires pour les extractions de données de bloc-objet mobile. Ici encore, la commande nécessaire de bus vient des dispositifs 6566 et 6567.

INTERFACE DE MÉMOIRE

Les versions 6566 et 6567 de la microplaquette d'interface vidéo ont des configurations de sortie d'adresse différentes. La 6566 possède 13 adresses totalement décodées pour la connexion directe au bus d'adresse du système. La 6567 possède des adresses multiplexées pour la connexion directe aux mémoires RAM dynamiques de 64 K. Les bits d'adresse les moins significatifs, A06—A00, se trouvent sur A06—A00 quand RAS/ est à l'état bas; les bits les plus significatifs, A13—A08, se trouvent sur A05—A00 quand CAS/ est à l'état bas. Les broches A11—A07 de la 6567 sont des sorties d'adresses statiques qui permettent la connexion directe de ces bits à une mémoire morte conventionnelle de 16 K (2 K x 8). (Un verrouillage externe est nécessaire pour les adresses de poids faible.)

INTERFACE D'UNITÉ DE TRAITEMENT

En dehors des accès spéciaux de mémoire décrits ci-dessus, on peut accéder aux registres des 6566 et 6567 comme à tout autre dispositif périphérique. On dispose des signaux suivants d'interface d'unité de traitement:

BUS DE DONNÉES (DB7—DB0)

Les huit broches de données correspondent à l'accès de données bidirectionnel, commandé par CS/, R/W et la phase 0. On ne peut accéder au bus de données que si la commande de validation d'adresse et la phase 0 sont à l'état haut et CS/ à l'état bas.

SÉLECTION DE MICROPLAQUETTE (CS/)

La broche de sélection de microplaquette CS/ est amenée à l'état bas pour permettre l'accès aux registres du dispositif, de concert avec les broches d'adresse et R/W. CS/ à l'état bas n'est accepté que si la commande de validation d'adresse et la phase 0 sont à l'état haut.

LECTURE/ÉCRITURE (R/W)

L'entrée de lecture/écriture (R/W) sert à déterminer la direction de transfert des données sur le bus de données, de concert avec CS/. Quand R/W est à l'état haut ("1"), les données sont transférées du registre choisi à la sortie de bus de données. Quand R/W est à l'état bas ("0"), les données présentées aux broches du bus de données sont chargées dans le registre choisi.

BUS D'ADRESSE (A05—A00)

Les six broches d'adresse inférieures A5—A0 sont bidirectionnelles. Pendant une lecture ou une écriture du dispositif vidéo par l'unité de traitement, ces broches d'adresse

correspondent à des entrées. Les données aux entrées d'adresse choisissent le registre de lecture ou d'écriture, défini dans la topographie de registre.

SORTIE D'HORLOGE (PHO)

La sortie d'horloge ou phase 0 correspond à l'horloge de 1 MHz servant d'entrée de phase 0 à l'unité de traitement 65XX. Toute l'activité du bus du système se rapporte à cette horloge. On obtient la fréquence d'horloge en divisant l'horloge d'entrée vidéo de 8 MHz par huit.

INTERRUPTIONS (IRQ/)

La sortie d'interruption IRQ/ est mise à l'état bas quand une source validée d'interruption se produit dans le dispositif. La sortie IRQ/ est un drain ouvert qui doit être équipé d'une résistance de charge extérieure.

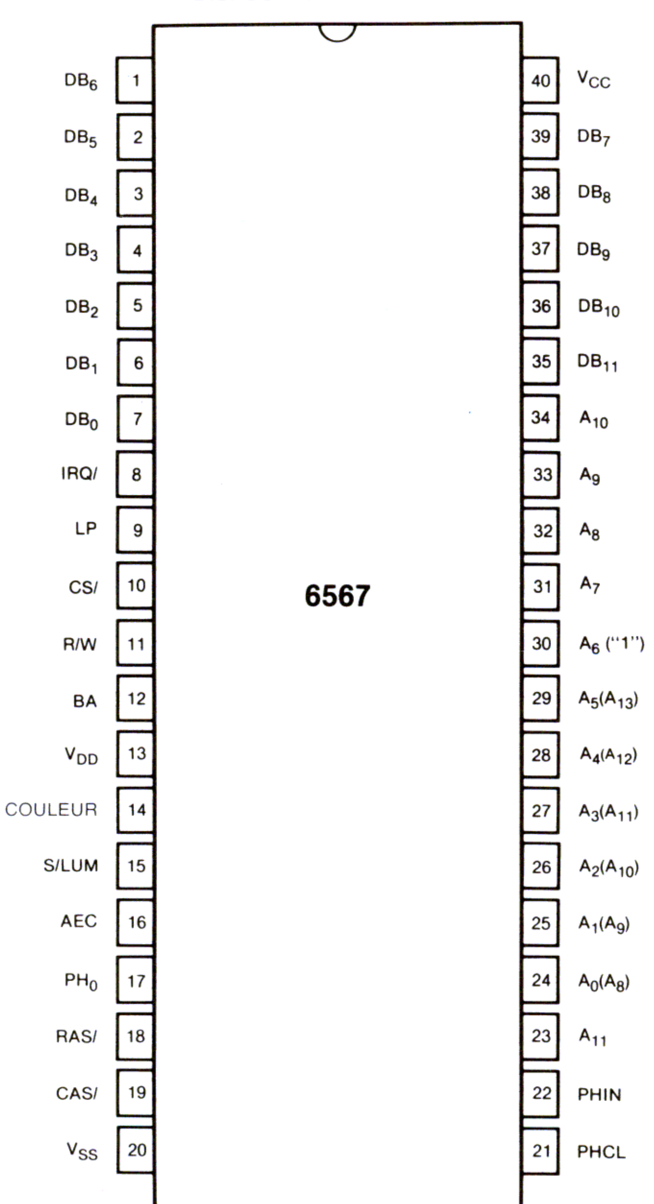
INTERFACE VIDÉO

Le signal de sortie vidéo de la 6566 ou 6567 se répartit en deux signaux qui doivent être extérieurement mélangés. La sortie SYNC/LUM contient toutes les données vidéo, y compris les synchros horizontale et verticale ainsi que les informations de luminance de l'affichage vidéo. La sortie SYNC/LUM est un drain ouvert qui doit être équipé d'une résistance de charge extérieure de 500 ohms. La sortie couleur comprend toutes les informations de chrominance y compris l'impulsion de synchronisation de sous-porteuse et la couleur de toutes les données d'affichage. La sortie de couleur est une source ouverte qui doit se terminer par une résistance de 1000 ohms reliée à la masse. Après le mélange approprié de ces deux signaux, on peut injecter directement le signal résultant dans un contrôleur vidéo ou dans un modulateur pour l'utilisation avec un téléviseur ordinaire.

RÉSUMÉ DE L'ACTIVITÉ DU BUS DE 6566/6567

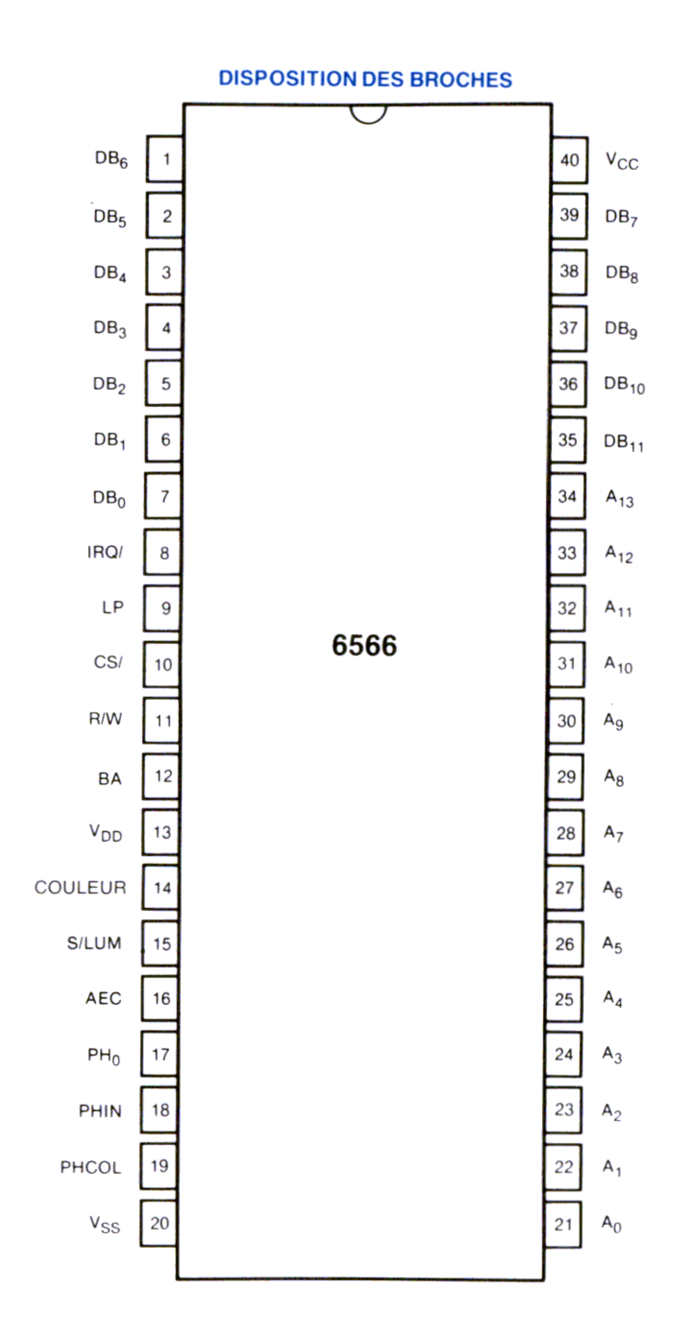
AEC	PHO	CS/	R/W	OPÉRATION
0	0	X	X	EXTRACTION, RÉGÉNÉRATION PHASE 1
0	1	X	X	EXTRACTION PHASE 2 (UNITÉ DE TRAITEMENT COUPÉE)
1	0	X	X	AUCUNE ACTION
1	1	0	0	ÉCRITURE AU REGISTRE CHOISI
1	1	0	1	LECTURE DU REGISTRE CHOISI
1	1	1	X	AUCUNE ACTION

DISPOSITION DES BROCHES



Les adresses multiplexées sont entre parenthèses

DISPOSITION DES BROCHES



TOPOGRAPHIE DES REGISTRES

ADRESSE		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPTION
00	($00)	M0X7	M0X6	M0X5	M0X4	M0X3	M0X2	M0X1	M0X0	Pos. X de bloc-objet mobile 0
01	($01)	M0Y7	M0Y6	M0Y5	M0Y4	M0Y3	M0Y2	M0Y1	M0Y0	Pos. Y de bloc-objet mobile 0
02	($02)	M1X7	M1X6	M1X5	M1X4	M1X3	M1X2	M1X1	M1X0	Pos. X de bloc-objet mobile 1
03	($03)	M1Y7	M1Y6	M1Y5	M1Y4	M1Y3	M1Y2	M1Y1	M1Y0	Pos. Y de bloc-objet mobile 1
04	($04)	M2X7	M2X6	M2X5	M2X4	M2X3	M2X2	M2X1	M2X0	Pos. X de bloc-objet mobile 2
05	($05)	M2Y7	M2Y6	M2Y5	M2Y4	M2Y3	M2Y2	M2Y1	M2Y0	Pos. Y de bloc-objet mobile 2
06	($06)	M3X7	M3X6	M3X5	M3X4	M3X3	M3X2	M3X1	M3X0	Pos. X de bloc-objet mobile 3
07	($07)	M3Y7	M3Y6	M3Y5	M3Y4	M3Y3	M3Y2	M3Y1	M3Y0	Pos. Y de bloc-objet mobile 3
08	($08)	M4X7	M4X6	M4X5	M4X4	M4X3	M4X2	M4X1	M4X0	Pos. X de bloc-objet mobile 4
09	($09)	M4Y7	M4Y6	M4Y5	M4Y4	M4Y3	M4Y2	M4Y1	M4Y0	Pos. Y de bloc-objet mobile 4
10	($0A)	M5X7	M5X6	M5X5	M5X4	M5X3	M5X2	M5X1	M5X0	Pos. X de bloc-objet mobile 5
11	($0B)	M5Y7	M5Y6	M5Y5	M5Y4	M5Y3	M5Y2	M5Y1	M5Y0	Pos. Y de bloc-objet mobile 5
12	($0C)	M6X7	M6X6	M6X5	M6X4	M6X3	M6X2	M6X1	M6X0	Pos. X de bloc-objet mobile 6
13	($0D)	M6Y7	M6Y6	M6Y5	M6Y4	M6Y3	M6Y2	M6Y1	M6Y0	Pos. Y de bloc-objet mobile 6
14	($0E)	M7X7	M7X6	M7X5	M7X4	M7X3	M7X2	M7X1	M7X0	Pos. X de bloc-objet mobile 7
15	($0F)	M7Y7	M7Y6	M7Y5	M7Y4	M7Y3	M7Y2	M7Y1	M6Y0	Pos. Y de bloc-objet mobile 7
16	($10)	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	Bit le plus sign. de la pos. X
17	($11)	RC8	ECM	BMM	DEN	RSEL	Y2	Y1	Y0	Voir texte
18	($12)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Registre de trame
19	($13)	LPX8	LPX7	LPX6	LPX5	LPX4	LPX3	LPX2	LPX1	X pour crayon lumineux
20	($14)	LPY7	LPY6	LPY5	LPY4	LPY3	LPY2	LPY1	LPY0	Y pour crayon lumineux
21	($15)	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	Validation bloc-objet mobile
22	($16)	—	—	RES	MCM	CSEL	X2	X1	X0	Voir texte
23	($17)	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	Agrand. Y de bloc-objet mobile

24	($18)	VM13	VM12	VM11	VM10	CB13	CB12	CB11	—	Pointeurs de mémoire
25	($19)	IRQ	—	—	—	ILP	IMMC	IMBC	IRST	Registre d'interruptions
26	($1A)	—	—	—	—	ELP	EMMC	EMBC	ERST	Interruption de validation
27	($1B)	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	Priorité car. objet mobile-données
28	($1C)	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	Sélect. multicolore bloc-objet mobile
29	($1D)	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	Agrand. X de bloc-objet mobile
30	($1E)	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	Collision de blocs-objets mobiles
31	($1F)	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	Collision bloc-objet/ données
32	($20)	—	—	—	—	EC3	EC2	EC1	EC0	Couleur extérieure
33	($21)	—	—	—	—	B0C3	B0C2	B0C1	B0C0	Couleur d'arrière-plan n° 0
34	($22)	—	—	—	—	B1C3	B1C2	B1C1	B1C0	Couleur d'arrière-plan n° 1
35	($23)	—	—	—	—	B2C3	B2C2	B2C1	B2C0	Couleur d'arrière-plan n° 2
36	($24)	—	—	—	—	B3C3	B3C2	B3C1	B3C0	Couleur d'arrière-plan n° 3
37	($25)	—	—	—	—	MM03	MM02	MM01	MM00	Bloc-objet multicolore n° 0
38	($26)	—	—	—	—	MM13	MM12	MM11	MM10	Bloc-objet multicolore n° 1
39	($27)	—	—	—	—	M0C3	M0C2	M0C1	M0C0	Coul. de bloc-objet mobile 0
40	($28)	—	—	—	—	M1C3	M1C2	M1C1	M1C0	Coul. de bloc-objet mobile 1
41	($29)	—	—	—	—	M2C3	M2C2	M2C1	M2C0	Coul. de bloc-objet mobile 2
42	($2A)	—	—	—	—	M3C3	M3C2	M3C1	M3C0	Coul. de bloc-objet mobile 3
43	($2B)	—	—	—	—	M4C3	M4C2	M4C1	M4C0	Coul. de bloc-objet mobile 4
44	($2C)	—	—	—	—	M5C3	M5C2	M5C1	M5C0	Coul. de bloc-objet mobile 5
45	($2D)	—	—	—	—	M6C3	M6C2	M6C1	M6C0	Coul. de bloc-objet mobile 6
46	($2E)	—	—	—	—	M7C3	M7C2	M7C1	M7C0	Coul. de bloc-objet mobile 7

REMARQUE: Un tiret indique l'absence de connexion. La lecture d'une absence de connexion correspond à "1".

CODES DE COULEUR

D4	D3	D1	D0	HEX.	DÉC.	COULEUR
0	0	0	0	0	0	NOIR
0	0	0	1	1	1	BLANC
0	0	1	0	2	2	ROUGE
0	0	1	1	3	3	TURQUOISE
0	1	0	0	4	4	VIOLET
0	1	0	1	5	5	VERT
0	1	1	0	6	6	BLEU
0	1	1	1	7	7	JAUNE
1	0	0	0	8	8	ORANGE
1	0	0	1	9	9	BRUN
1	0	1	0	A	10	ROUGE CLAIR
1	0	1	1	B	11	GRIS FONCÉ
1	1	0	0	C	12	GRIS MOYEN
1	1	0	1	D	13	VERT CLAIR
1	1	1	0	E	14	BLEU CLAIR
1	1	1	1	F	15	GRIS CLAIR

Annexe O

SPÉCIFICATIONS DE LA MICROPLAQUETTE DE DISPOSITIF D'INTERFACE DE SON (SID) 6581

CONCEPT

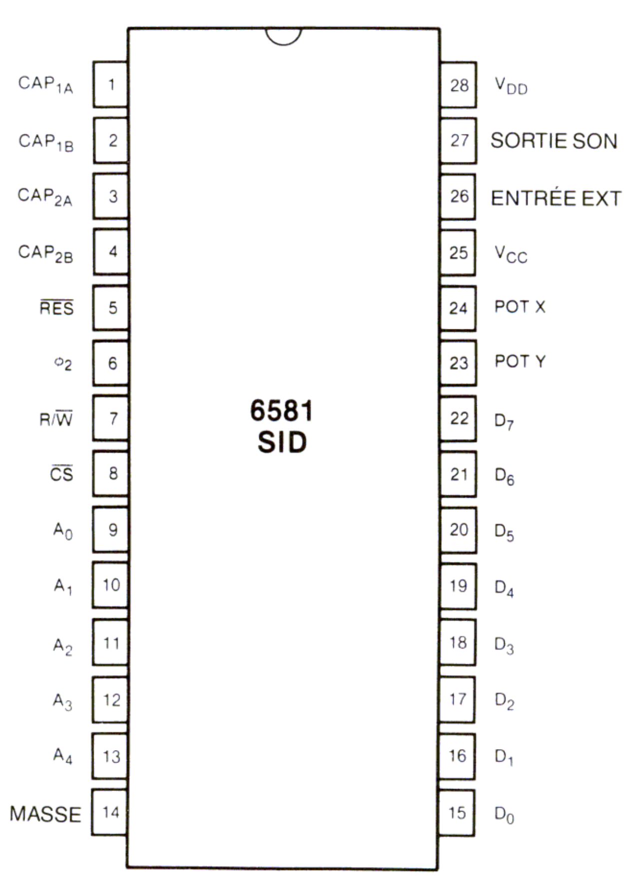
Le dispositif d'interface de son (SID) 6581 est un générateur d'effets sonores/synthétiseur de musique électronique à trois voix sur une microplaquette compatible avec les familles de microprocesseurs 65XX et les familles similaires. Le dispositif d'interface de son permet le contrôle à haute définition et dans une large gamme, de la hauteur du son (fréquence), du timbre (teneur en harmoniques) et de l'élément dynamique (volume). Des circuits spécialisés de commande réduisent au minimum le temps du système de logiciel pour faciliter la création de jeux électroniques et d'instruments de musique économiques.

CARACTÉRISTIQUES

- 3 OSCILLATEURS DE TONALITÉ
  - Gamme: 0—4 kHz
- 4 FORMES D'ONDE PAR OSCILLATEUR
  - Triangulaire, dents de scie, impulsion variable, bruit
- 3 MODULATEURS D'AMPLITUDE
  - Gamme: 48 dB
- 3 GÉNÉRATEURS D'ENVELOPPE
  - Réponse exponentielle
  - Régime d'attaque: 2 ms—8 s
  - Régime de décroissance: 6 ms—24 s
  - Niveau de stabilisation: 0— volume maximal
  - Régime d'extinction: 6 ms—24 s
- SYNCHRONISATION DES OSCILLATEURS
- MODULATION DIRECTIONNELLE

- FILTRE PROGRAMMABLE
  - Intervalle de coupure: 30 Hz à 12 kHz
  - Pente de 12 dB/octave
  - Sorties passe-bas, bande passante, passe-haut et de créneau
  - Résonnance variable
- COMMANDE PRINCIPALE DE VOLUME
- DEUX INTERFACES DE POTENTIOMÈTRE ANALOGIQUE/NUMÉRIQUE
- GÉNÉRATEUR DE NOMBRES ALÉATOIRES/MODULATION
- ENTRÉE SON EXTÉRIEURE

DISPOSITION DES BROCHES



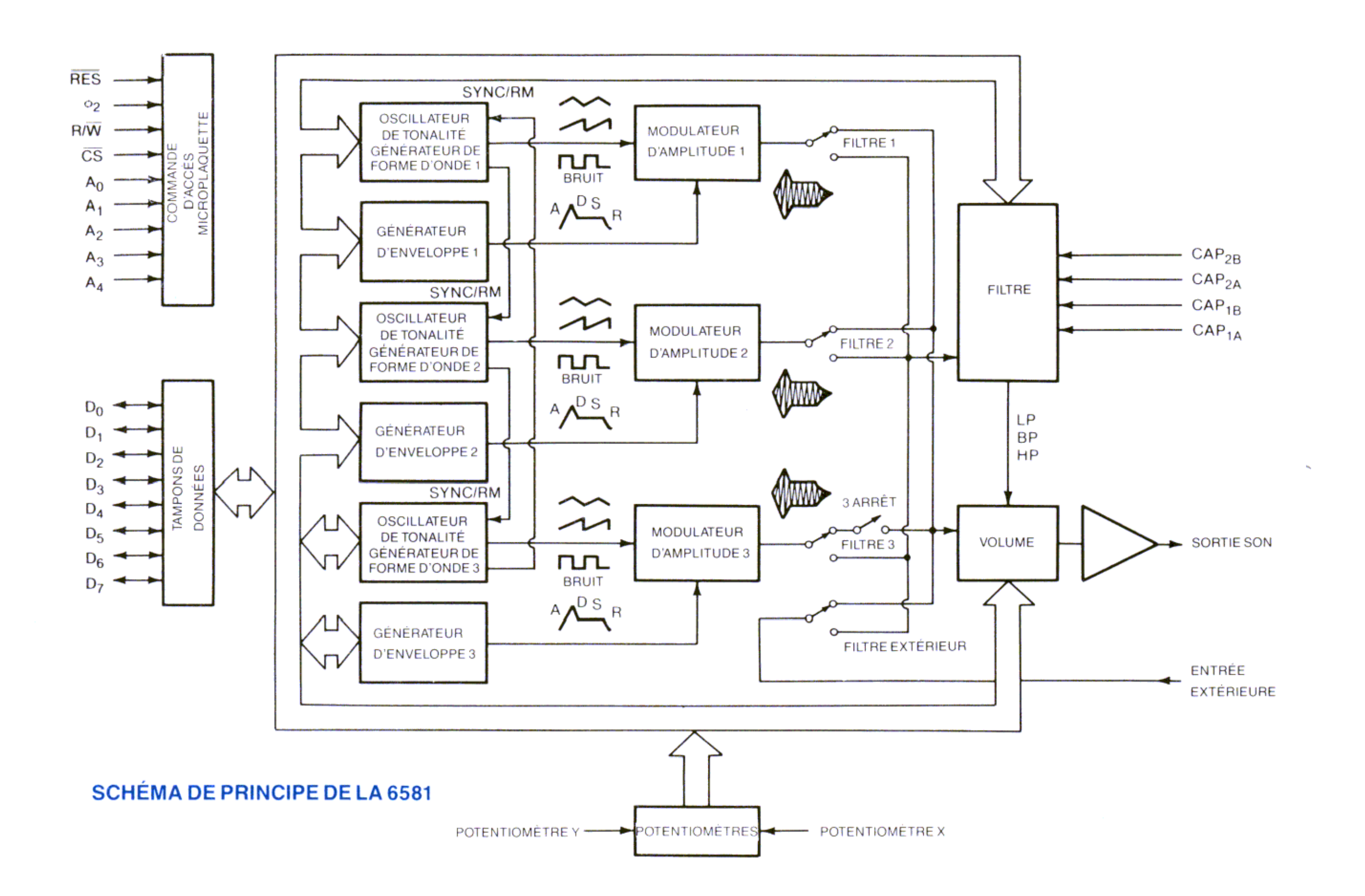


SCHÉMA DE PRINCIPE DE LA 6581

DESCRIPTION

La 6581 comprend trois "voix" de synthétiseur que l'on peut utiliser séparément ou ensemble (ou avec des sources sonores extérieures) pour créer des sons complexes. Chaque voix se compose d'un oscillateur de tonalité/générateur de forme d'onde, d'un générateur d'enveloppe et d'un modulateur d'amplitude. L'oscillateur de tonalité commande la hauteur de la voix dans une gamme large. L'oscillateur produit quatre formes d'onde à la fréquence choisie, avec la teneur unique en harmoniques de chacune d'elles permettant la commande simple du timbre. L'élément dynamique du volume de l'oscillateur est commandé par le modulateur d'amplitude, par l'intermédiaire du générateur d'enveloppe. Quand il est déclenché, le générateur d'enveloppe crée une enveloppe d'amplitude avec des régimes programmables de volume en hausse et en baisse. En plus des trois voix, un filtre programmable permet de créer des timbres dynamiques complexes par synthèse soustractive.

La microplaquette de dispositif d'interface de son (SID) permet la lecture par le microprocesseur de la sortie variable du troisième oscillateur et du troisième générateur d'enveloppe. On peut utiliser ces sorties comme source d'information de modulation pour créer des balayages de fréquence/filtrage, des effets de vibrato, etc. Le troisième oscillateur peut aussi servir de générateur de nombres aléatoires pour les jeux. Deux convertisseurs analogiques/numériques permettent l'interface de la microplaquette SID avec des potentiomètres; ceux-ci peuvent servir de "palettes" dans les jeux ou de commandes avec un synthétiseur de musique. La microplaquette SID peut traiter les signaux sonores extérieurs. Il est possible de monter plusieurs microplaquettes SID en guirlande ou de les mélanger dans des systèmes polyphoniques complexes.

REGISTRES DE COMMANDE DE MICROPLAQUETTE SID

La microplaquette SID contient 29 registres de 8 bits qui commandent la création du son. Ces registres sont de type à écriture ou à lecture seulement. Ils sont indiqués dans la table 1 ci-dessous.

Table 1: Topographie des registres de SID

	ADRESSE					REG. N°	DONNÉES								NOM DU REG.	TYPE DU REG.
	A_4	A_3	A_2	A_1	A_0	(HEX)	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
															Voix 1	
0	0	0	0	0	0	00	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	Fréquence basse	Écriture seulement
1	0	0	0	0	1	01	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	Fréquence haute	Écriture seulement
2	0	0	0	1	0	02	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	Imp. basse	Écriture seulement
3	0	0	0	1	1	03	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	Imp. haute	Écriture seulement
4	0	0	1	0	0	04	Bruit	⊓⊔⊓	⟋|⟋|	⋀⋀	Contrôle	Mod. dir.	SYNC	Porte	Registre de commande	Écriture seulement
5	0	0	1	0	1	05	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	Attaque/décroissance	Écriture seulement
6	0	0	1	1	0	06	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	Stabilisation/extinction	Écriture seulement
															Voix 2	
7	0	0	1	1	1	07	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	Fréquence basse	Écriture seulement
8	0	1	0	0	0	08	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	Fréquence haute	Écriture seulement
9	0	1	0	0	1	09	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	Imp. basse	Écriture seulement
10	0	1	0	1	0	0A	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	Imp. haute	Écriture seulement
11	0	1	0	1	1	0B	Bruit	⊓⊔⊓	⟋|⟋|	⋀⋀	Contrôle	Mod. dir.	SYNC	Porte	Registre de commande	Écriture seulement
12	0	1	1	0	0	0C	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	Attaque/décroissance	Écriture seulement
13	0	1	1	0	1	0D	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	Stabilisation/extinction	Écriture seulement
															Voix 3	
14	0	1	1	1	0	0E	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	Fréquence basse	Écriture seulement
15	0	1	1	1	1	0F	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	Fréquence haute	Écriture seulement
16	1	0	0	0	0	10	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	Imp. basse	Écriture seulement
17	1	0	0	0	1	11	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	Imp. haute	Écriture seulement
18	1	0	0	1	0	12	Bruit	⊓⊔⊓	⟋|⟋|	⋀⋀	Contrôle	Mod. dir.	SYNC	Porte	Registre de commande	Écriture seulement
19	1	0	0	1	1	13	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	Attaque/décroissance	Écriture seulement
20	1	0	1	0	0	14	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	Stabilisation/extinction	Écriture seulement
															Filtre	
21	1	0	1	0	1	15	—	—	—	—	—	FC_2	FC_1	FC_0	Coupure filtre bas	Écriture seulement
22	1	0	1	1	0	16	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3	Coupure filtre haut	Écriture seulement
23	1	0	1	1	1	17	RES_3	RES_2	RES_1	RES_0	FILTEX	FILT 3	FILT 2	FILT 1	Remise état initial/filtre	Écriture seulement
24	1	1	0	0	0	18	3 Arrêt	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0	Mode/volume	Écriture seulement
															Divers	
25	1	1	0	0	1	19	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0	Potentiomètre X	Lecture seulement
26	1	1	0	1	0	1A	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	Potentiomètre Y	Lecture seulement
27	1	1	0	1	1	1B	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	Oscillateur 3/aléatoire	Lecture seulement
28	1	1	1	0	0	1C	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	Enveloppe 3	Lecture seulement

DESCRIPTION DES REGISTRES DE SID

VOIX 1

FRÉQUENCE BASSE/FRÉQUENCE HAUTE (registres 00,01)

Ces registres réunis forment un nombre de 16 bits qui commande linéairement la fréquence de l'oscillateur 1. La fréquence est déterminée par l'équation suivante:

$$F_{sortie} = (F_n \times F_{horl.}/16777216)\ Hz$$

Dans laquelle F_n correspond au nombre de 16 bits des registres de fréquence et $F_{horl.}$ à l'horloge de système appliquée à l'entrée $\phi 2$ (broche 6). Pour une horloge standard de 1 MHz, la fréquence est donnée par:

$$F_{sortie} = (F_n \times 0.059604645)\ Hz$$

L'annexe E donne une table complète des valeurs de création de 8 octaves de la gamme musicale uniformément tempérée avec le la (440 Hz). Il faut remarquer que la résolution de fréquence de la microplaquette SID suffit à toute gamme d'accord et permet l'enchaînement d'une note à l'autre (glissade) sans décalage de fréquence notable.

IMPULSION BASSE/IMPULSION HAUTE (registres 02,03)

Ces registres réunis forment un nombre de 12 bits (les bits 4 à 7 d'impulsion haute ne sont pas utilisés) qui commande linéairement la largeur d'impulsion (cycle de service) de la forme d'onde d'impulsion à l'oscillateur 1. La largeur d'impulsion est déterminée par l'équation suivante:

$$L\ imp = (L\ impn/40.95)\%$$

dans laquelle L impn correspond au nombre de 12 bits dans les registres de largeur d'impulsion.

La résolution de largeur d'impulsion permet l'enchaînement en douceur sans gradation notable. Il faut remarquer que l'on doit choisir la forme d'onde d'impulsion à l'oscillateur 1 de façon que les registres de largeur d'impulsion aient un effet audible quelconque. Une valeur de zéro ou de 4095 ($FF) dans les registres de largeur d'impulsion donne une sortie CC constante; une valeur de 2048 ($800) donne une onde carrée.

REGISTRE DE COMMANDE (registre 04)

Ce registre contient 8 bits de commande qui choisissent les différentes options à l'oscillateur 1.

PORTE 1 (bit 0): Le bit de porte commande le générateur d'enveloppe pour la voix 1. Quand ce bit est fixé à 1, le générateur d'enveloppe est déclenché et le cycle d'ATTAQUE/DÉCROISSANCE/STABILISATION est lancé. Quand le bit est remis à zéro, le cycle d'EXTINCTION commence. Le générateur d'enveloppe commande l'amplitude de l'oscillateur 1 qui apparaît à la sortie son; de ce fait, le bit de porte doit être à l'état 1 (avec les paramètres d'enveloppe corrects) pour que la sortie choisie de l'oscillateur 1 soit audible. On trouvera une étude détaillée du générateur d'enveloppe à la fin de cette annexe.

SYNCHRO (bit 1): Le bit de synchro, quand il est à 1, synchronise les fréquences fondamentales de l'oscillateur 1 avec celles de l'oscillateur 3 pour créer des effets de "synchro fixe".

La variation de la fréquence de l'oscillateur 1 par rapport à celle de l'oscillateur 3 donne une grande diversité de structures harmoniques complexes de la voix 1 à la fréquence de l'oscillateur 3. Pour que la synchro se fasse, l'oscillateur 3 doit être fixé à une fréquence autre que zéro et de préférence inférieure à celle de l'oscillateur 1. Aucun autre paramètre de voix 3 n'a d'effet sur la synchro.

MODULATEUR DIRECTIONNEL (bit 2): Le bit de modulateur directionnel, quand il est fixé à 1, remplace la sortie de forme d'onde triangulaire de l'oscillateur 1 par une combinaison à "modulation directionnelle" des oscillateurs 1 et 3. La variation de la fréquence de l'oscillateur 1 par rapport à celle de l'oscillateur 3 donne une gamme étendue de structures non harmoniques pour des sons de gongs ou de cloches et des effets spéciaux. Pour que la modulation directionnelle soit audible, la forme d'onde triangulaire de l'oscillateur 1 doit être choisie et l'oscillateur 3 doit être fixé à une fréquence différente de zéro. Aucun autre paramètre de voix 3 n'a d'effet sur la modulation directionnelle.

CONTROLE (bit 3): Le bit de contrôle, s'il est fixé à 1, remet l'oscillateur 1 à zéro et le verrouille jusqu'à l'effacement de ce bit. La sortie de forme d'onde de bruit de l'oscillateur 1 est aussi remise à zéro et la sortie de forme d'onde d'impulsion est maintenue à un niveau CC. On utilise normalement ce bit pour le contrôle, mais il peut aussi servir à synchroniser l'oscillateur 1 pour des opérations extérieures; il permet la création de formes d'onde très complexes sous contrôle du logiciel en temps réel.

(Bit 4): Quand ce bit est à 1, la sortie de forme d'onde triangulaire de l'oscillateur 1 est choisie. La forme d'onde triangulaire, qui contient peu d'harmoniques, donne une douce sonorité de flûte.

(Bit 5): Quand ce bit est à 1, la sortie de forme d'onde en dents de scie de l'oscillateur 1 est choisie. La forme d'onde en dents de scie donne une tonalité riche en harmoniques paires et impaires; son timbre est éclatant et claironnant.

(Bit 6): Quand ce bit est à 1, la sortie de forme d'onde d'impulsion de l'oscillateur 1 est choisie. On peut régler la teneur en harmoniques de cette forme d'onde à l'aide des registres de largeur d'impulsion pour obtenir des tonalités allant de l'onde carrée caverneuse et vive à l'impulsion grêle et nasillarde. L'étalement de la largeur d'impulsion produit un effet de "mise en phase" dynamique qui apporte une sensation de mobilité au son. Le passage rapide entre différentes largeurs d'impulsion peut donner des séquences harmoniques intéressantes.

BRUIT (bit 7): Quand ce bit est à 1, la sortie de forme d'onde de bruit de l'oscillateur 1 est choisie. Cette sortie donne un signal aléatoire qui varie suivant la fréquence de l'oscillateur 1. On peut faire varier la qualité sonore du bourdonnement grave jusqu'au sifflement de bruit blanc par l'intermédiaire des registres de fréquence de l'oscillateur 1. Cette forme d'onde sert à créer des bruits d'explosion, d'arme à feu, d'avion à réaction, de vent, de ressac et d'autres sons sans variation de timbre, ainsi que les cymbales et les caisses claires. Le balayage de la fréquence d'oscillateur avec le bruit choisi produit un effet de souffle intéressant.

On doit choisir l'une des formes d'onde de sortie pour que l'oscillateur 1 soit audible; toutefois, il n'est PAS utile de couper les formes d'onde pour arrêter la sortie de la voix 1. L'amplitude de la voix 1 à la sortie finale dépend uniquement du générateur d'enveloppe.

Remarque: Les formes d'onde de sortie d'oscillateur ne s'ajoutent PAS. Si l'on choisit simultanément plus d'une forme d'onde de sortie, il en résulte une opération logique ET sur ces formes d'onde. On peut utiliser cette technique pour créer d'autres formes d'onde en dehors des quatre indiquées ci-dessus, mais il faut procéder avec précaution. Si l'on choisit une autre forme d'onde quand le bruit est en fonction, la sortie de bruit peut se "bloquer". Dans ce cas, la sortie bruit reste coupée jusqu'à ce qu'elle soit rétablie par le bit de contrôle ou en mettant RES (broche 5) à l'état bas.

ATTAQUE/DÉCROISSANCE (registre 05)

Les bits 4 à 7 de ce registre (ATK0 à ATK3) choisissent un des 16 régimes d'attaque pour le générateur d'enveloppe de voix 1. Le régime d'attaque détermine la rapidité de montée de la sortie de la voix 1, de 0 à l'amplitude de crête quand le générateur d'enveloppe est déclenché. La table 2 donne les 16 régimes d'attaque.

Les bits 0 à 3 (DCY0 à DCY3) choisissent un des 16 régimes de décroissance pour le générateur d'enveloppe. Le cycle de décroissance suit le cycle d'attaque et le régime de décroissance détermine la rapidité de chute de la sortie, de l'amplitude de crête au niveau choisi de stabilisation. La table 2 donne les 16 régimes de décroissance.

STABILISATION/EXTINCTION (registre 06)

Les bits 4 à 7 de ce registre (STN0 à STN3) choisissent un des 16 niveaux de stabilisation pour le générateur d'enveloppe. Le cycle de stabilisation suit le cycle de décroissance; la sortie de la voix 1 reste à l'amplitude choisie de stabilisation tant que le bit de porte reste à 1. Les niveaux de stabilisation s'étendent de 0 à l'amplitude de crête, en 16 degrés linéaires. Une valeur de stabilisation de 0 correspond à une amplitude nulle et une valeur de 15 ($F) donne l'amplitude de crête. Une valeur de stabilisation de 8 amène la voix 1 à se stabiliser à une amplitude correspondant à la moitié de l'amplitude de crête atteinte par le cycle d'attaque.

Les bits 0 à 3 (RLS0 à RLS3) choisissent un des 16 régimes d'extinction pour le générateur d'enveloppe. Le cycle d'extinction suit le cycle de stabilisation quand le bit de porte est remis à 0. À ce moment, la sortie de la voix 1 tombe de l'amplitude de stabilisation à l'amplitude 0, au régime d'extinction choisi. Les 16 régimes d'extinction sont identiques aux régimes de décroissance.

REMARQUE: On peut modifier en tout point le cycle du générateur d'enveloppe à l'aide du bit de porte. Le générateur d'enveloppe peut être déclenché et relâché sans limitation. Par exemple, si le bit de porte est remis à zéro avant que l'enveloppe ait terminé le cycle d'attaque, le cycle d'extinction commence immédiatement à l'amplitude qui a été atteinte. Si l'enveloppe est ensuite déclenchée de nouveau (avant que le cycle d'extinction soit arrivé à l'amplitude zéro), un autre cycle d'attaque commence, à l'amplitude qui a été atteinte. On peut utiliser cette technique pour créer des enveloppes d'amplitude complexes sous contrôle du logiciel en temps réel.

Table 2. Régimes d'enveloppe

VALEUR		RÉGIME D'ATTAQUE	RÉGIME DE DÉCROIS-SANCE/EXTINCTION
DÉC.	(HEX.)	(Durée/cycle)	(Durée/cycle)
0	(0)	2 ms	6 ms
1	(1)	8 ms	24 ms
2	(2)	16 ms	48 ms
3	(3)	24 ms	72 ms
4	(4)	38 ms	114 ms
5	(5)	56 ms	168 ms
6	(6)	68 ms	204 ms
7	(7)	80 ms	240 ms
8	(8)	100 ms	300 ms
9	(9)	250 ms	750 ms
10	(A)	500 ms	1.5 s
11	(B)	800 ms	2.4 s
12	(C)	1 s	3 s
13	(D)	3 s	9 s
14	(E)	5 s	15 s
15	(F)	8 s	24 s

REMARQUE: Les régimes d'enveloppe sont basés sur une horloge φ 2 de 1 MHz. Pour les autres fréquences φ 2, multiplier le régime donné par 1MHz/φ2. Les régimes se rapportent à la durée par cycle. Par exemple, avec une valeur d'attaque de 2, le cycle d'attaque prend 16 ms pour monter de 0 à l'amplitude de crête. Le régime de décroissance/extinction se rapporte à la durée nécessaire à ces cycles pour tomber de l'amplitude de crête à zéro.

VOIX 2

Les registres 07 à $0D commandent la voix 2. Sur le plan fonctionnel, ils sont identiques aux registres 00 à 06 avec les exceptions suivantes:

1) S'il est choisi, le registre de synchro synchronise l'oscillateur 2 avec l'oscillateur 1.
2) S'il est choisi, le registre de modulation directionnelle remplace la sortie triangulaire de l'oscillateur 2 par la combinaison de modulation directionnelle des oscillateurs 2 et 1.

VOIX 3

Les registres $0E à $14 commandent la voix 3. Sur le plan fonctionnel, ils sont identiques aux registres 00 à 06, avec les exceptions suivantes:

1) S'il est choisi, le registre de synchro synchronise l'oscillateur 3 avec l'oscillateur 2.
2) S'il est choisi, le registre de modulation directionnelle remplace la sortie triangulaire de l'oscillateur 3 par la combinaison de modulation directionnelle des oscillateurs 3 et 2.

Le fonctionnement typique d'une voix consiste à choisir les paramètres désirés: fréquence, forme d'onde, effets (synchro, modulation directionnelle) et régime d'enveloppe; on déclenche ensuite la voix quand on désire obtenir le son. On peut stabiliser le son pendant toute durée et le terminer en effaçant le bit de porte. On peut utiliser séparément chaque voix, avec paramètres et déclenchement indépendants ou simultanément pour créer une seule voix de grande puissance. Si les voix sont utilisées simultanément, un léger désaccord de chaque oscillateur ou l'accord sur les intervalles musicaux crée une sonorité riche et vivante.

Filtre

FRÉQUENCE DE COUPURE BASSE/FRÉQUENCE DE COUPURE HAUTE (registres $15,$16)

Ces registres réunis forment un nombre de 11 bits (les bits 3 à 7 du registre de fréquence basse ne sont pas utilisés) qui commande linéairement la fréquence de coupure (centrale) du filtre programmable. La fréquence de coupure approximative s'étend de 30 Hz à 12 kHz.

RÉSONNANCE/FILTRAGE (registre $17)

Les bits 4 à 7 de ce registre (RES0 à RES3) commandent la résonnance du filtre. La résonnance du filtre est un effet de crête qui accentue les composantes de fréquence à la fréquence de coupure du filtre pour donner un son plus pénétrant. Les 16 réglages de résonnance s'étendent linéairement de la résonnance nulle (0) à la résonnance maximale (15 ou $F). Les bits 0 à 3 déterminent les signaux acheminés par le filtre.

FILTRE 1 (Bit 0): Si ce bit est à zéro, la voix 1 apparaît directement à la sortie son et n'est pas affectée par le filtre. Si ce bit est à 1, la voix 1 est traitée par le filtre et sa teneur en harmoniques est modifiée suivant les paramètres choisis du filtre.

FILTRE 2 (Bit 1): Identique au bit 0, pour la voix 2.

FILTRE 3 (Bit 2): Identique au bit 0, pour la voix 3.

FILTRE EXTÉRIEUR (Bit 3): Identique au bit 0, pour l'entrée son extérieur (broche 26).

MODE/VOLUME (registre $18)

Les bits 4 à 7 de ce registre choisissent différentes options de sortie et de mode de filtre:

PASSE-BAS (LP) (bit 4): Quand ce bit est à 1, la sortie passe-bas du filtre est choisie et envoyée à la sortie son. Avec un signal d'entrée de filtre donné, toutes les composantes de fréquence au-dessous de la fréquence de coupure du filtre passent intégralement et toutes les composantes au-dessus de la coupure sont atténuées à raison de 12 dB par octave. Le mode de filtre passe-bas donne des sons de grande ampleur.

BANDE PASSANTE (BP) (bit 5): Identique au bit 4, mais pour la sortie de bande passante. Toutes les composantes de fréquence au-dessus et au-dessous de la coupure sont atténuées à raison de 6 dB par octave. Le mode de bande passante donne des sons légers et clairs.

PASSE-HAUT (HP) (bit 6): Identique au bit 4, mais pour la sortie passe-haut. Toutes les composantes de fréquence au-dessus de la coupure passent intégralement et toutes les composantes au-dessous de la coupure sont atténuées à raison de 12 dB par octave. Le mode passe-haut donne des sons bourdonnants et métalliques.

3 ARRÊT (bit 7): Quand ce bit est à 1, la sortie de la voix 3 est coupée du trajet son direct. Si l'on règle la voix 3 pour qu'elle ne passe pas par le filtre (FILT 3 = 0) et si on met 3 arrêt (3 OFF) à 1, la voix 3 ne peut pas arriver à la sortie son. On peut ainsi utiliser la voix 3 pour la modulation sans sortie indésirable.

REMARQUE: Les modes de sortie de filtre s'ajoutent. On peut choisir simultanément des modes multiples de filtre. Par exemple, on peut choisir les modes LP et HP ensemble pour produire un filtre de créneau (ou de rejet de bande). Pour que le filtre ait un effet audible, on doit choisir au moins une sortie de filtre et faire passer une voix au moins par le filtre. Le filtre est probablement l'élément le plus important de la microplaquette SID, car il permet la création de tonalités complexes par synthèse soustractive (le filtre élimine les composantes spécifiques de fréquence d'un signal d'entrée riche en harmoniques). On obtient les meilleurs résultats en faisant varier la fréquence de coupure en temps réel.

Bits 0 à 3 (VOL0 à VOL3) choisissent un des 16 niveaux de volume d'ensemble pour la sortie son mixte finale. Les niveaux de volume de sortie s'étendent de la sortie nulle (0) au volume maximal (15 ou $F) en 16 degrés linéaires. On peut utiliser cette commande de volume statique pour équilibrer les niveaux dans les systèmes à plusieurs microplaquettes ou pour créer des effets de volume dynamique, comme le trémolo. On doit choisir un niveau de volume différent de 0 pour la SID produise un son.

DIVERS

POTX (registre $19)

Ce registre permet au microprocesseur de lire la position du potentiomètre relié à POTX (broche 24) avec des valeurs s'étendant de 0, à la résistance minimale, à 255 ($FF), à la résistance maximale. La valeur est toujours valide et est mise à jour tous les 512 cycles d'horloge ϕ 2. Pour plus de détails sur les valeurs de potentiomètre et de condensateur, voir la section sur la description des broches.

POTY (registre $1A)

Identique à POTX pour le potentiomètre relié à POTY (broche 23).

OSCILLATEUR 3/NOMBRE ALÉATOIRE (registre $1B)

Ce registre permet au microprocesseur de lire les 8 bits de sortie supérieurs de l'oscillateur 3. Le caractère des nombres créés se rapporte directement à la forme d'onde choisie. Si l'on choisit la forme d'onde en dents de scie de l'oscillateur 3, ce registre présente une série de nombres augmentant de 0 à 255 ($FF) à un régime déterminé par la fréquence de l'oscillateur 3. Si l'on choisit la forme d'onde triangulaire, la sortie augmente de 0 et 255 puis diminue jusqu'à 0. Si l'on choisit la forme d'onde d'impulsion, la sortie saute entre 0 et 255. Si l'on choisit la forme d'onde de bruit, on obtient une série de nombres aléatoires; on peut ainsi utiliser ce registre comme générateur de nombres aléatoires pour les jeux. Il existe de nombreuses applications de synchronisation et de mise en séquence pour le registre d'oscillateur 3; toutefois, celui-ci remplit essentiellement la fonction d'un générateur de modulation. Les nombres créés par ce registre peuvent être additionnés, sous logiciel, aux registres de fréquence d'oscillateur ou de filtre ou aux registres de largeur d'impulsion, en temps réel. On peut ainsi créer de nombreux effets dynamiques. On peut produire des sons de sirène en ajoutant la sortie de dents de scie de l'oscillateur 3 à la commande de fréquence d'un autre oscillateur. On peut produire des effets "d'échantillonnage-blocage" de synthétiseur en ajoutant la sortie bruit de l'oscillateur 3 aux registres de commande de fréquence de filtre. On peut obtenir un effet de vibrato en réglant l'oscillateur 3 à une fréquence d'environ 7 Hz et en ajoutant la sortie triangulaire de l'oscillateur 3 (avec cadrage convenable) à la commande de fréquence d'un autre oscillateur. On dispose d'une gamme illimitée d'effets en modifiant la fréquence de l'oscillateur 3 et en cadrant sa sortie. Dans des conditions normales, quand on utilise l'oscillateur 3 pour la modulation, on doit éliminer la sortie son de la voix 3 (3 ARRÊT = 1).

ENVELOPPE 3 (registre $1C)

Ce registre est identique à l'oscillateur 3, mais il permet au microprocesseur de lire la sortie du générateur d'enveloppe de voix 3. On peut ajouter cette sortie à la fréquence de filtre pour obtenir des enveloppes harmoniques, des aboiements ou des effets analogues. On peut créer des sons de cadrage en ajoutant cette sortie au registre de commande de fréquence d'un oscillateur. Le générateur d'enveloppe de voix 3 doit être déclenché pour produire une sortie à partir de ce registre. Toutefois, le registre d'oscillateur 3 reflète toujours la sortie variable de l'oscillateur et n'est pas affecté par le générateur d'enveloppe.

DESCRIPTION DES BROCHES DE LA SID

CAP1A, CAP1B, (broches 1,2)/CAP2A, CAP2B (broches 3,4)

Ces broches servent à relier les deux condensateurs d'intégrateur requis par le filtre programmable. C1 se branche entre les broches 1 et 2 et C2 entre les broches 3 et 4. Les deux condensateurs doivent avoir la même valeur. Dans la gamme sonore, de 30 Hz à 12 kHz environ, le filtre fonctionne normalement avec des condensateurs C1 et C2 de 2200 pF. Il est préférable d'utiliser des condensateurs en polystyrène et, dans les systèmes polyphoniques complexes où plusieurs microplaquettes SID sont alignées les unes sur les autres, on recommande des condensateurs équilibrés.

On peut adapter la gamme de fréquences du filtre à des applications particulières par le choix des valeurs des condensateurs. Par exemple, une excellente réponse aux fréquences élevées peut n'être pas nécessaire pour un jeu économique. Dans ce cas, on peut choisir des valeurs plus élevées de C1 et de C2 pour disposer d'un meilleur contrôle des fréquences graves du filtre. La fréquence de coupure maximale du filtre est donnée par:

$FC_{max} = 2.6E-5/C$

Dans laquelle C représente la valeur des condensateurs. La gamme du filtre s'étale sur 9 octaves au-dessous de la fréquence de coupure maximale.

RES (Remise à l'état initial) (broche 5)

Cette entrée de niveau TTL correspond à la commande de remise à zéro de la microplaquette SID. Quand cette entrée est à l'état bas pendant un minimum de 10 cycles $\phi 2$, tous les registres internes sont remis à zéro et la sortie son est coupée. Cette broche est normalement reliée à la ligne de remise à zéro du microprocesseur ou d'un circuit d'effacement à la mise sous tension.

$\phi 2$ (broche 6)

Cette entrée de niveau TTL constitue l'horloge principale de la microplaquette SID. Toutes les fréquences d'oscillateur et tous les régimes d'enveloppe sont rattachés à cette horloge. $\phi 2$ commande aussi les transferts de données entre la microplaquette SID et le microprocesseur. Le transfert des données ne peut se faire que si $\phi 2$ est à l'état haut. $\phi 2$ sert essentiellement de sélection de microplaquette active à l'état haut pour le transfert de données. Cette broche est normalement reliée à l'horloge du système, avec une fréquence d'exploitation nominale de 1 MHz.

R/W (Lecture/écriture) (broche 7)

Cette entrée de niveau TTL commande le sens de transfert des données entre la microplaquette SID et le microprocesseur. Si les conditions de sélection de microplaquette sont satisfaites, un état haut sur cette ligne permet au microprocesseur de lire (R) les données du registre choisi de la SID; un état bas permet au microprocesseur d'écrire (W) des données dans le registre choisi de la microplaquette SID. Cette broche est normalement reliée à la ligne de lecture/écriture (R/W) du système.

CS (Sélection de microplaquette) (broche 8)

Cette entrée de niveau TTL correspond à une sélection de microplaquette active à l'état bas qui commande le transfert des données entre la microplaquette SID et le microprocesseur. CS doit être à l'état bas pour tout transfert de données. Une lecture du registre choisi de microplaquette SID ne peut se faire que si CS est à l'état bas et si ϕ 2 et R/W sont à l'état haut. Une écriture au registre choisi de la microplaquette SID ne peut se produire que si CS est à l'état bas, ϕ 2 à l'état haut et R/W à l'état bas. Cette broche, normalement reliée au circuit de décodage d'adresse, permet à la SID de résider dans la topographie de mémoire d'un système.

A0 à A4 (broches 9 à 13)

Ces entrées de niveau TTL servent à choisir l'un des 29 registres de la SID. Il existe assez d'adresses pour choisir l'un des 32 registres, mais les trois positions restantes de registre ne sont pas utilisées. Toute écriture dans l'une de ces trois positions est ignorée. Une lecture se traduit par le retour de données invalides. Ces broches, normalement reliées aux lignes d'adresses correspondantes du microprocesseur permettent d'adresser la microplaquette SID de la même manière que la mémoire.

GND (Masse) (broche 14)

Pour la qualité des résultats, la ligne de masse de la microplaquette SID et l'alimentation doivent être séparées des lignes de masse des autres circuits numériques. On minimise ainsi le bruit numérique à la sortie son.

D0 à D7 (broches 15 à 22)

Ces lignes bidirectionnelles servent au transfert des données entre la microplaquette SID et le microprocesseur. Elles sont compatibles TTL en mode d'entrée et capables de commander 2 charges TTL en mode de sortie. Les tampons de données sont généralement à l'arrêt à haute impédance. Pendant une opération d'écriture, les tampons de données restent à l'état d'arrêt (entrée) et le microprocesseur fournit les données à la microplaquette SID par ces lignes. Pendant une opération de lecture, les tampons de données se mettent en fonction et la microplaquette SID envoie les données au microprocesseur par ces lignes. Ces broches sont normalement reliées aux lignes de données correspondantes du microprocesseur.

POTX,POTY (Potentiomètre X, potentiomètre Y) (broches 24,23)

Ces broches correspondent aux entrées des convertisseurs analogiques/numériques qui servent à numériser la position des potentiomètres. Le processus de conversion s'appuie sur la constante de temps d'un condensateur, entre la broche POT et la masse, qui est chargé par un potentiomètre relié entre la broche POT et le +5 volts. Les valeurs des composantes sont déterminées par:

$$RC = 4.7E-4$$

dans laquelle R représente la résistance maximale du potentiomètre et C le condensateur.

L'instabilité de la valeur POT est inversement proportionnelle à la capacité du condensateur. On recommande les valeurs de 470 kilohms et de 1000 pF pour R et C. Remarquer qu'il faut un potentiomètre et un condensateur séparés pour chaque broche POT.

Vcc (broche 25)

Comme pour la ligne de masse (GND), on doit avoir une ligne CC +5 V séparée entre Vcc de la microplaquette SID et l'alimentation pour minimiser le bruit. Monter un condensateur de découplage près de la broche.

EXT IN (Entrée extérieure) (broche 26)

Cette entrée analogique permet le mélange des signaux sonores extérieurs à la sortie son de la plaquette SID qui peut avoir été traitée dans le filtre. Parmi les sources typiques, citons la voix, la guitare et l'orgue. L'impédance d'entrée de cette broche est de l'or-

dre de 100 kilohms. Tout signal appliqué directement à cette broche doit avoir un niveau CC de 6 volts et ne pas dépasser 3 volts crête à crête. Pour éviter toute interférence provoquée par les différences de niveau CC, les signaux extérieurs doivent être reliés par couplage alternatif à l'entrée extérieure (EXT IN) par l'intermédiaire d'un condensateur électrolytique de 1 à 10 μF. Le trajet son direct (FILTEX = 0) ayant un gain unitaire, on peut utiliser l'entrée extérieure pour mélanger les sorties de plusieurs microplaquettes SID montées en guirlande. Le nombre de microplaquettes pouvant être ainsi montées en guirlande dépend de l'importance de la distorsion et du bruit permis à la sortie finale. Noter que la commande de volume de sortie affecte les sorties extérieures, en plus des trois voix de la microplaquette SID.

(Sortie son) (broche 27)

Ce tampon à source ouverte correspond à la sortie son finale de la microplaquette SID qui comprend les trois voix de la SID, le filtre et toute entrée extérieure. Le niveau de sortie, qui est fixé par le volume de commande de sortie, atteint un maximum de 2 volts crête à crête à un niveau CC de 6 volts. Pour le bon fonctionnement, on doit prévoir une résistance de source entre la sortie audio et la masse. La résistance recommandée est de 1 kilohm pour une impédance de sortie standard.

La sortie de la microplaquette SID étant à un niveau CC de 6 volts, on doit la relier, par couplage alternatif, à un amplificateur de son à l'aide d'un condensateur électrolytique de 1 à 10 μF.

V_{DD} (broche 28)

Comme pour V_{CC}, on doit prévoir une ligne CC + 12 V séparée reliée à V_{DD} de la microplaquette SID; utiliser un condensateur de découplage.

CARACTÉRISTIQUES DE LA MICROPLAQUETTE SID 6581

VALEURS MAXIMALES ABSOLUES

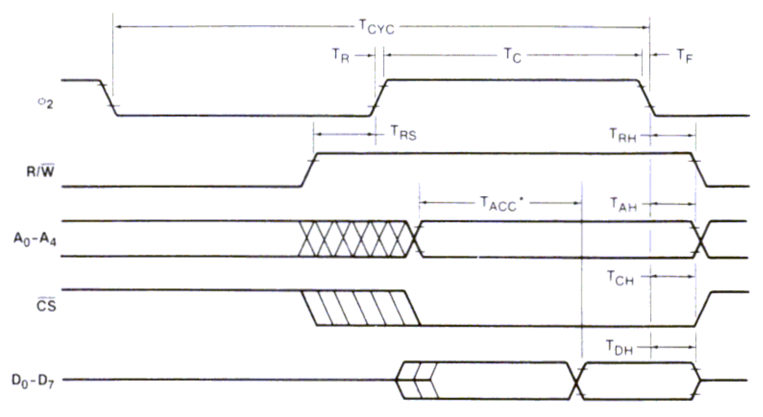
VALEUR	SYMBOLE	VALEUR	UNITÉ
Tension d'alimentation	V_{DD}	−0.3 à +17	V CC
Tension d'alimentation	V_{CC}	−0.3 à +7	V CC
Tension d'entrée (analogique)	V_{ina}	−0.3 à +17	V CC
Tension d'entrée (numérique)	V_{ind}	−0.3 à +7	V CC
Température de fonctionnement	T_A	0 à +70	°C
Température de remisage	T_{STG}	−55 à +150	°C

CARACTÉRISTIQUES ÉLECTRIQUES (VDD = 12 V CC ±5%, VCC = 5 V CC ±5%, TA = 0 à 70°C)

CARACTÉRISTIQUE		SYMBOLE	MIN.	TYP.	MAX.	UNITÉ
Tension haute d'entrée	(RES,ϕ2, R/W, CS,	V_{IH}	2	—	V_{CC}	V CC
Tension basse d'entrée	A0–A4, D0–D7)	V_{IL}	−0.3	—	0.8	V CC
Courant de fuite d'entrée	(RES,ϕ2,R/W, CS, A0–A4; V_{in}=0–5 V CC)	I_{in}	—	—	2.5	μA
Trois états (arrêt)	(D0–D7; V_{CC}=max.)	I_{TSI}	—	—	10	μA
Courant de fuite d'entrée	V_{in}=0.4–2.4 V CC					
Tension haute de sortie	(D0–D7; V_{CC}=min., I charge=200 μA)	V_{OH}	2.4	—	$V_{CC}-0.7$	V CC
Tension basse de sortie	(D0–D7; V_{CC}=max., I charge=3.2 mA)	V_{OL}	MASSE	—	0.4	V CC
Courant haut de sortie	(D0–D7; source, V_{OH}=2.4 V CC)	I_{OH}	200	—	—	μA

Courant bas de sortie	(D0–D7; débit, V_{OL} = 0.4 V CC)	I_{OL}	3.2	—	—	mA
Capacité d'entrée	(RES, $\phi 2$, R/W, CS, A0–A4, D0–D7)	C_{in}	—	—	10	pF
Tension de déclenchement de potentiomètre	(POTX, POTY)	V_{pot}	—	$V_{CC}/2$	—	V CC
Courant de chute de potentiomètre	(POTX, POTY)	I_{pot}	500	—	—	μA
Impédance d'entrée	(Ent. ext.)	R_{in}	100	150	—	kΩ
Tension d'entrée son	(Ent. ext.)	V_{in}	5.7 —	6 0.5	6.3 3	V CC V CA
Tension de sortie son	(Sortie son; 1 kΩ charge, volume = max.) Une voix en fonction: Toutes les voix en fonction:	V_{out}	5.7 0.4 1.0	6 0.5 1.5	6.3 0.6 2.0	V CC V CA V CA
Courant d'alimentation	(V_{DD})	I_{DD}	—	20	25	mA
Courant d'alimentation	(V_{CC})	I_{CC}	—	70	100	mA
Dissipation de puissance	(totale)	P_D	—	600	1000	mW

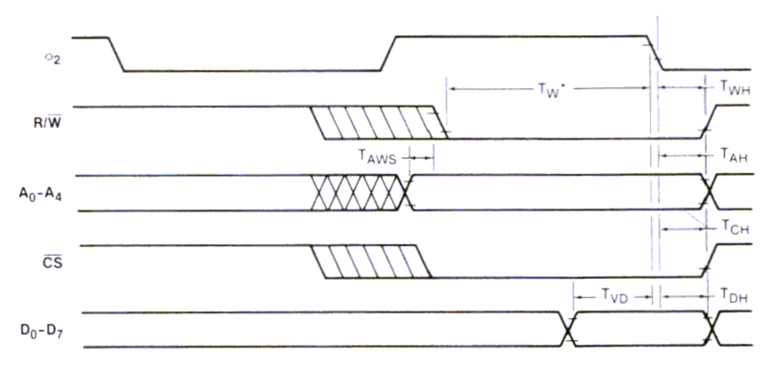
SYNCHRONISATION DE SID 6581



*T_{ACC} est mesurée après ϕ_2, CS ou A_0-A_4, suivant l'action se produisant en dernier.

CYCLE DE LECTURE (READ)

SYMBOLE	NOM	MIN.	TYP.	MAX.	UNITÉ
T_{CYC}	Temps de cycle d'horloge	1	—	20	µs
T_C	Largeur d'impulsion haute d'horloge	450	500	10,000	ns
T_R, T_F	Durée de montée/chute d'horloge	—	—	25	ns
T_{RS}	Durée d'établissement de lecture	0	—	—	ns
T_{RH}	Durée de maintien de lecture	0	—	—	ns
T_{ACC}	Durée d'accès	—	—	300	ns
T_{AH}	Durée de maintien d'adresse	10	—	—	ns
T_{CH}	Durée de maintien de sélection de microplaquette	0	—	—	ns
T_{DH}	Durée de maintien de données	20	—	—	ns



*T_W est mesurée à partir de ϕ_2, CS ou R/W, suivant l'action se produisant en dernier.

CYCLE D'ÉCRITURE

SYMBOLE	NOM	MIN.	TYP.	MAX.	UNITÉ
T_W	Largeur d'impulsion d'écriture	300	—	—	ns
T_{WH}	Durée de maintien d'écriture	0	—	—	ns
T_{AWS}	Durée d'établissement d'adresse	0	—	—	ns
T_{AH}	Durée de maintien d'adresse	10	—	—	ns
T_{CH}	Durée de maintien de sélection de microplaquette	0	—	—	ns
T_{VD}	Données valides	80	—	—	ns
T_{DH}	Temps de maintien des données	10	—	—	ns

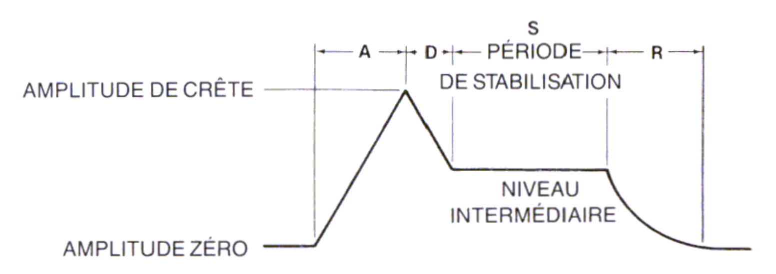
VALEURS DE LA GAMME MUSICALE UNIFORMÉMENT TEMPÉRÉE

Le tableau de l'annexe E donne les valeurs numériques à mémoriser dans les registres de commande de fréquence d'oscillateur de la microplaquette SID pour obtenir les notes de la gamme musicale uniformément tempérée. La gamme uniformément tempérée comprend une octave se composant de 12 demi-tons (notes): do, ré, mi, fa, sol, la, si, do #, ré #, fa #, sol # et la #. La fréquence de chaque demi-ton correspond exactement à la racine douzième de ($\sqrt[12]{2}$) par la fréquence du demi-ton précédent. La table est basée sur une horloge $\phi 2$ de 1.02 MHz. Voir l'équation indiquée dans la description des registres pour l'utilisation d'autres fréquences d'horloge principale. La gamme choisie correspond à la hauteur pour concert, avec A-4 = 440 Hz. Il est également possible de procéder à des transpositions de cette gamme et d'autres que la gamme uniformément tempérée.

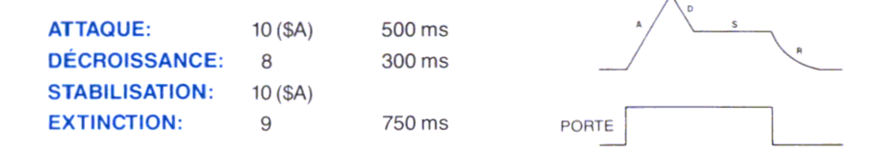
La table de l'annexe E donne une méthode simple et rapide de création de la gamme uniformément tempérée, mais elle est très peu efficace sur le plan de la mémoire car il faut 192 octets pour la table seule. On peut améliorer l'efficacité de la mémoire en déterminant la valeur des notes par algorithme. Si l'on considère que chaque note dans une octave correspond exactement à la moitié de la fréquence de cette note dans l'octave suivante, on peut ramener la table de référence de 96 à 12 introductions, car il y a douze notes par octave. Si les 12 introductions (24 octets) se composent des valeurs à 16 bits de la huitième octave (do-7 à ré-7), on peut alors en tirer les notes des octaves plus basses en choisissant la note correspondante de la huitième octave et en divisant la valeur à 16 bits par 2 pour chaque octave de différence. Une division par deux correspond en fait en un décalage à droite de la valeur; un simple programme de routine en logiciel permet de faire facilement ce calcul. La note ré-7 dépasse la gamme des oscillateurs, mais cette valeur doit encore être comprise dans la table aux fins de calcul (le bit le plus significatif de ré-7 est un cas spécial de logiciel; on peut créer ce bit dans le report avant le décalage). Spécifier chaque note sous une forme indiquant le demi-ton désiré parmi les 12 existants et celle des huit octaves ou se trouve le demi-ton. Quatre bits étant nécessaires au choix de l'un des 12 demi-tons et trois bits étant requis pour choisir l'un des huit octaves, on peut loger les informations dans un octet, avec le demi-octet de poids faible déterminant le demi-ton (avec adressage à la table de référence) et le demi-octet de poids fort servant au programme de routine de division pour déterminer le nombre de décalages à droite de la valeur de la table.

GÉNÉRATEURS D'ENVELOPPE DE LA SID

En musique électronique, le générateur d'enveloppe en quatre parties (attaque, décroissance, stabilisation et extinction) s'est avéré le meilleur compromis entre diversité d'emploi et facilité de commande de l'amplitude. Le choix convenable des paramètres d'enveloppe permet la simulation d'un grand nombre d'instruments à percussion et à cordes. Le violon est un excellent exemple d'instrument à cordes. Le violoniste commande l'amplitude du son à l'aide de l'archet. En général, l'amplitude augmente lentement, atteint une crête, puis baisse jusqu'à un niveau intermédiaire. Le violoniste peut maintenir ce niveau aussi longtemps qu'il le désire et le laisser lentement tomber. Nous donnons ci-dessous un croquis de cette enveloppe:



On peut facilement reproduire cette enveloppe d'amplitude, à attaque/décroissance/stabilisation/extinction, avec les valeurs typiques indiquées ci-dessous:



ATTAQUE:	10 ($A)	500 ms
DÉCROISSANCE:	8	300 ms
STABILISATION:	10 ($A)	
EXTINCTION:	9	750 ms

Noter que l'on peut maintenir le son au niveau de stabilisation intermédiaire aussi longtemps qu'on le désire. Le son ne commence à s'estomper qu'à la disparition de la porte. On peut utiliser cette enveloppe de base, avec des modifications mineures, pour les instruments à vent, les cuivres et les instruments à cordes.

On obtient une forme d'enveloppe totalement différente avec les instruments à percussion comme les tambours, les cymbales et les gongs ainsi qu'avec certains instruments à clavier comme le piano et le clavecin. L'enveloppe d'une percussion se caractérise par une attaque presque instantanée suivie immédiatement d'une décroissance jusqu'à 0.

Les instruments à percussion ne se stabilisent pas à une amplitude constante. Par exemple, quelle que soit la manière dont on frappe un tambour, le son atteint instantanément l'amplitude maximale et décroît rapidement. Nous indiquons ci-dessous une enveloppe de cymbales:

ATTAQUE:	0	2 ms
DÉCROISSANCE:	9	750 ms
STABILISATION:	0	
EXTINCTION:	9	750 ms

On peut remarquer que le son commence immédiatement à décroître jusqu'à zéro après avoir atteint la crête, même si la porte ne disparaît pas. L'enveloppe d'amplitude des pianos et clavecins est nettement plus compliquée mais on peut la créer très facilement avec l'attaque/décroissance/stabilisation/extinction. Ces instruments arrivent à l'amplitude maximale au moment où on frappe une touche. L'amplitude commence immédiatement à s'estomper lentement tant que l'on maintient le doigt sur la touche. Si l'on relâche la touche avant que le son se soit complètement estompé, l'amplitude descend immédiatement à zéro. Cette enveloppe a l'aspect suivant:

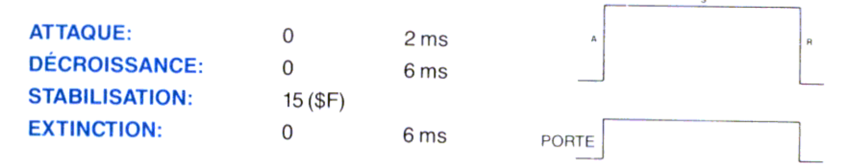
ATTAQUE:	0	2 ms
DÉCROISSANCE:	9	750 ms
STABILISATION:	0	
EXTINCTION:	0	6 ms



On peut remarquer que le son décroît lentement jusqu'à la disparition de la porte; à ce point, l'amplitude descend rapidement à zéro.

L'orgue présente l'enveloppe la plus simple. Quand on appuie sur une touche, le son atteint immédiatement l'amplitude maximale et y reste. Quand on relâche la touche, l'amplitude descend immédiatement à zéro. Nous donnons cette enveloppe ci-dessous:

ATTAQUE:	0	2 ms
DÉCROISSANCE:	0	6 ms
STABILISATION:	15 ($F)	
EXTINCTION:	0	6 ms

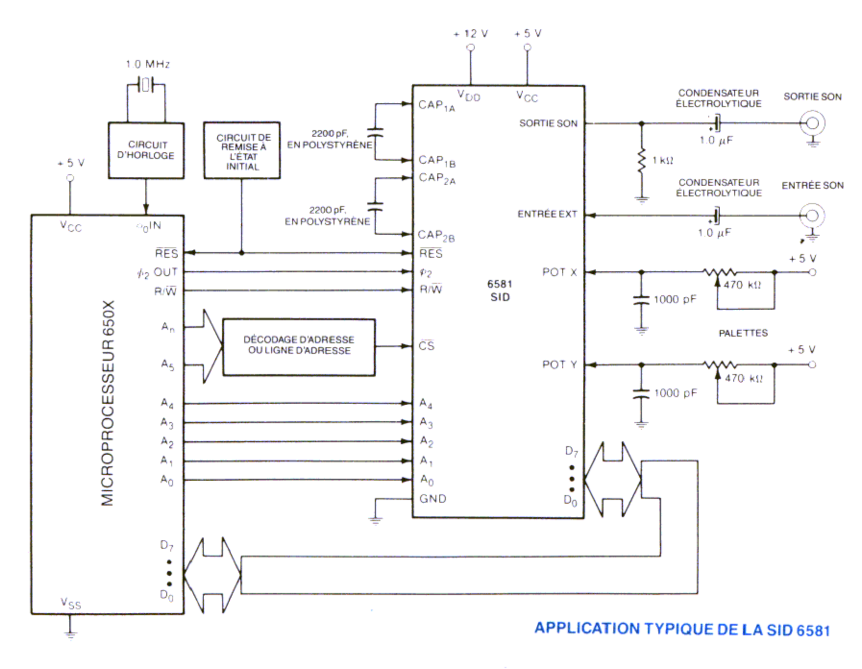


La grande qualité de la microplaquette SID réside dans sa capacité à créer des sons originaux plutôt qu'à imiter des instruments de musique. L'attaque/ décroissance/stabilisation/extinction permet de créer des enveloppes qui ne correspondent à aucun instrument véritable. À cet égard, mentionnons l'enveloppe "inverse". Cette enveloppe, caractérisée par une attaque lente et une décroissance rapide, ressemble de près à un instrument enregistré sur une bande que l'on reproduirait à l'envers. Nous donnons cette enveloppe ci-dessous:

ATTAQUE:	10 ($A)	500 ms
DÉCROISSANCE:	0	6 ms
STABILISATION:	15 ($F)	
EXTINCTION:	3	72 ms

S
A
R
PORTE

On peut créer de nombreux sons originaux en appliquant l'enveloppe d'amplitude d'un instrument à la structure harmonique d'un autre. On obtient ainsi des sons qui, tout en étant analogues à ceux d'instruments de musique familiers, en diffèrent notablement. En général, le son est de nature très subjective; il faut procéder à des essais avec différents contenus en harmoniques et régimes d'enveloppe pour arriver au son désiré.



APPLICATION TYPIQUE DE LA SID 6581

ANNEXE P

GLOSSAIRE

Attaque	Vitesse avec laquelle une note de musique atteint son amplitude de crête
Binaire	Système de numération à base 2
Bruit chromatique	Distorsion de couleur
CIA	Adaptateur d'interface complexe
Décimal	Système de numération à base 10
Décroissance	Vitesse avec laquelle une note de musique descend de l'amplitude de crête à l'amplitude de stabilisation
DDR	Registre de direction de données
e	Constante mathématique (environ 2.71828183)
Écran vidéo	Téléviseur
Élément d'image	Point de définition sur l'écran
Entier	Nombre entier, sans point décimal
Enveloppe	Forme de l'amplitude d'une note en fonction du temps.
Extinction	Vitesse avec laquelle une note de musique descend de l'amplitude de stabilisation à l'amplitude zéro.
FIFO	Premier entré/premier sorti
File	Ligne simple
Hexadécimal	Système de numération à base 16
Horloge d'heure	Minuterie d'intervalle de la machine
Indice inférieur	Variable d'index
NMI	Interruption non invalidable
Nombres avec signe	Nombres positifs ou négatifs
Octal	Système de numération à base 8
Octet	Position de mémoire
Opérande	Paramètre
Opérateurs booléens	Opérateurs logiques
OS	Système d'exploitation
Registre	Position spéciale de stockage dans la mémoire
ROM	Mémoire morte
SID	Dispositif d'interface de son
Stabilisation	Amplitude de stabilisation d'une note de musique
Syntaxe	Structure d'une phrase de programmation
Tronqué	Coupé ou éliminé (et non arrondi)
VIC-II	Microplaquette d'interface vidéo

INDEX

CARTE DE RÉFÉRENCE RAPIDE DU COMMODORE 64

VARIABLES SIMPLES

Type	*Nom*	*Intervalle*
Réel	XY	±1.70141183E+38
		±2.93873588E−39
Entier	XY%	±32767
Chaîne	XY$	0 à 255 caractères

X est une lettre (A à Z), Y est une lettre ou un chiffre (0 à 9). Les noms de variable peuvent avoir plus de 2 caractères, mais les deux premiers seulement sont acceptés.

VARIABLES EN TABLEAU

Type	*Nom*
Une dimension	XY(5)
Deux dimensions	XY(5,5)
Trois dimensions	XY(5,5,5)

Le cas échéant, on peut utiliser des tableaux atteignant 11 éléments (indices inférieurs 0 à 10). Les tableaux ayant plus de 11 éléments doivent être dimensionnés (DIM).

OPÉRATEURS ALGÉBRIQUES

=	Attribue une valeur à une variable
−	Négation
↑	Élévation à une puissance
*	Multiplication
/	Division
+	Addition
−	Soustraction

OPÉRATEURS LOGIQUES ET DE RELATION

=	Égal
<>	N'est pas égal à
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
NON	"Non" logique
ET	"Et" logique
OU	"Ou" logique

L'expression est égale à 1 si elle est vraie et à 0 si elle est fausse.

COMMANDES DU SYSTÈME

LOAD "NOM"	Charge un programme depuis une cassette
SAVE "NOM"	Sauvegarde un programme sur cassette
LOAD "NOM", 8	Charge un programme à partir d'un disque
SAVE "NOM", 8	Sauvegarde un programme sur disque
VERIFY "NOM"	Vérifie que le programme a été sauvegardé (SAVE) sans erreur
RUN	Exécute un programme
RUN xxx	Exécute un programme à partir de la ligne xxx
STOP	Interrompt l'exécution
END	Termine l'exécution
CONT	Continue l'exécution du programme à partir de la ligne où le programme avait été interrompu
PEEK(X)	Retourne le contenu de la position de mémoire X
POKE X, Y	Change le contenu de la position X à la valeur Y
SYS xxxxx	Saute à l'exécution d'un programme en langage machine, à partir de xxxxx
WAIT X, Y, Z	Le programme attend jusqu'à ce que le contenu de la position X après une opération "OU exclusif" avec Z et une opération "ET" avec Y ne soit pas nul
USR(X)	Passe la valeur de X à un sous-programme en langage machine

COMMANDES D'ÉDITION ET DE MISE EN FORME

LIST	Liste le programme entier
LIST A-B	Liste le programme de la ligne A à la ligne B
REM Message	Message de commentaire pouvant être listé, mais ignoré pendant l'exécution du programme
TAB(X)	S'utilise dans les instructions PRINT. Espacement de X positions sur l'écran
SPC(X)	Imprime (PRINT) X espaces vierges sur la ligne
POS(X)	Retourne à la position présente du curseur
CLR/HOME	Positionne le curseur dans le coin gauche de l'écran
SHIFT CLR/HOME	Efface l'écran et place le curseur à la position de repos
SHIFT INST/DEL	Insère un espace à la position présente du curseur
INST/DEL	Annule le caractère à la position présente du curseur
CTRL	S'utilise avec une touche de couleur numérique, choisit la couleur du texte. Peut s'utiliser dans une instruction PRINT
Touches CRSR	Déplace le curseur vers le haut, le bas, la gauche ou la droite sur l'écran
Touche Commodore	Utilisée avec la touche SHIFT, permet de choisir entre majuscules/minuscules et caractères graphiques. Utilisée avec la touche de couleur numérique, choisit la couleur de texte facultative.

TABLEAUX ET CHAÎNES

DIM A(X,Y,Z)	Fixe les indices inférieurs maximaux pour A; réserve de la place pour (X+1)*(Y+1)*(Z+1) éléments à partir de A(0,0,0)
LEN (X$)	Retourne le nombre de caractères dans X
STR$(X)	Retourne la valeur numérique de X, convertie en chaîne
VAL(X$)	Retourne la valeur numérique de A$, jusqu'au premier caractère non numérique
CHR$(X)	Retourne le caractère ASCII dont le code est X
ASC(X$)	Retourne le code ASCII pour le premier caractère X$
LEFT(A$,X)	Retourne les X caractères les plus à gauche de la chaîne A$
RIGHT$(A$,X)	Retourne les X caractères les plus à droite de la chaîne A$
MID$(A$,X,Y)	Retourne Y caractères de la chaîne A$, à partir du caractère X

COMMANDES D'ENTRÉE/SORTIE

INPUT A$ OR A	Imprime (PRINT) "?" sur l'écran et attend que l'utilisateur introduise une chaîne ou une valeur
INPUT "ABC";A	Imprime (PRINT) un message et attend que l'utilisateur introduise une valeur. Peut aussi introduire A$
GET A$ or A	Attend que l'utilisateur tape une valeur d'un caractère; RETURN n'est pas nécessaire
DATA A,"B",C	Initialise un jeu de valeurs pouvant être utilisé par une instruction READ
READ A$ or A	Attribue la valeur DATA suivante à A$ ou A
RESTORE	Rétablit le pointeur de données pour commencer de nouveau la lecture (READ) de la liste DATA
PRINT "A= ";A	Imprime (PRINT) la chaîne "A= " et la valeur de A. ";" supprime les espaces— "," tabule les données à la zone suivante

DÉROULEMENT DU PROGRAMME

GOTO X	Branchement à la ligne X
IF A=3 THEN 10	Si (IF) la supposition est vraie, exécute donc (THEN) la partie suivante de l'instruction. Si la supposition est fausse, exécute le numéro de ligne suivant
FOR A=1 TO 10 STEP 2 : NEXT	Exécute toutes les instructions entre FOR et NEXT correspondant; A va de 1 à 10 par degrés de 2. Sauf mention contraire, les degrés varient de 1
NEXT A	Définit la fin de boucle. A est facultatif
GOSUB 2000	Branchement au sous-programme à partir de la ligne 2000
RETURN	Marque la fin du sous-programme. Revient à l'instruction suivant le GOSUB le plus récent
ON X GOTO A,B	Branchement au X[e] numéro de ligne de la liste. Si X = 1, branchement à A, etc.
ON X GOSUB A,B	Branchement au sous-programme au X[e] numéro de ligne de la liste

Abonnez-vous à ces magazines d'utilisateur pour tirer le maximum de votre ordinateur Commodore

POWER/PLAY   commodore

Loisirs, jeux et autres avec les ordinateurs individuels Commodore

Publié chaque trimestre en mars, juin, septembre et décembre, POWER/PLAY se consacre uniquement à l'univers en expansion rapide des ordinateurs individuels Commodore. Il donne de précieuses indications sur les nouveaux produits, les applications, les jeux, les techniques de programmation, l'enseignement progressif, les télécommunications et une multitude d'autres sujets pour que l'utilisateur d'un ordinateur individuel Commodore tire le maximum de sa machine. Abonnement: $10 par an.

Le magazine des micro-ordinateurs

Lue par les enseignants, les hommes d'affaires, les étudiants et les amateurs d'informatique, cette publication trimestrielle permet d'accéder à des renseignements exclusifs sur les systèmes Commodore, les techniques de programmation, les interfaces du matériel et les applications intéressant la vaste gamme des produits Commodore. Chaque numéro présente des articles intéressant l'utilisateur, présent ou futur, de l'équipement Commodore. Tirez le meilleur parti de votre micro-ordinateur, grâce à ce magazine. Abonnement: $15 par an.

PRIÈRE DE REMPLIR ET DE RENVOYER

Nom ____________________ Téléphone __________

Adresse ______________________________________

Ville __________ Province _____ Code postal _____

Modèle d'ordinateur: ____________________________

☐ Changement d'adresse. Indiquer la nouvelle adresse ci-dessus et joindre la présente étiquette d'expédition
☐ Réabonnement
☐ Nouvel abonnement

POUR EN AVOIR POUR VOTRE ARGENT

Veuillez m'abonner pour:

_____ an(s) à POWER/PLAY à $10 par an

_____ an(s) à COMMODORE à $15 par an

Canada et étranger: POWER/PLAY, $15 par an : COMMODORE, $25 par an

Ci-joint mon chèque ou mandat de $______________

Établir le chèque ou le mandat à l'ordre de:
COMMODORE BUSINESS MACHINES, INC.
The Meadows, 487 Devon Park Drive, Wayne, PA 19087

QUELQUES MOTS SUR LE GUIDE DE RÉFÉRENCE DU PROGRAMMEUR DE COMMODORE 64...

Comptabilité avec les cartouches de jeu. . .son spectaculaire. . .graphiques éclatants. . .possibilités de traitement remarquables. . .autant d'avantages qui font du Commodore 64 l'ordinateur individuel le plus évolué de sa catégorie pour l'usage personnel, les affaires et l'enseignement.

LE GUIDE DE RÉFÉRENCE DU PROGRAMMEUR DE COMMODORE 64 étudie en détail les possibilités du Commodore 64. Il complète parfaitement le guide de l'utilisateur du Commodore 64 avec ses explications approfondies allant des graphiques au son en passant par les techniques d'avant-garde de langage machine. Cet ouvrage est indispensable pour le débutant comme pour le programmeur chevronné.

Pour le débutant, les sujets les plus complexes sont présentés avec de nombreux exemples de programmes et un style d'une grande clarté. Pour le programmeur chevronné, cet ouvrage a subi de multiples contrôles visant à satisfaire vos besoins. Il est disposé de façon à tirer le maximum des possibilités étendues du Commodore 64.

ORDINATEURS
commodore
3370 Pharmacy Avenue,
Agincourt, Ontario M1W 2K4

IMPRIMÉ AU CANADA